UNIVERSITY OF AUGSBURG

DOCTORAL THESIS

Resilient Traffic Management

From reactive to proactive systems

Matthias SOMMER

A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Science

Chair of Organic Computing
Faculty of Computer Science

Advisor:	Prof. Dr. Jörg HÄHNER
Second advisor:	Prof. Dr. Sabine TIMPF
Date of oral exam:	06.12.2018

Abstract

This thesis explores ways to improve the self-organised urban traffic management system Organic Traffic Control by means of forecasting traffic developments and by applying machine learning techniques. Current traffic control systems typically rely on suboptimal human-designed signal plans which can not respond to changing traffic demands and become outdated over time. The goal of this thesis is to transform the reactive control cycle into an anticipatory and more resilient one. To achieve this goal, the observer/controller architecture for organic technical systems is extended with a forecast module for time series. Based on monitored sensor values, forecasts of the future traffic developments help to enrich the understanding of the complex dependencies within the traffic network. Furthermore, machine learning techniques are used to improve the performance of the Organic Traffic Control system at runtime.

In detail, the contributions of this thesis are as follows. To introduce resilience into technical systems, the reference design model for organic computing systems is extended by a module for forecasting of time series. This contribution enables the transformation of Organic Traffic Control from reactive to proactive adaptation. On the basis of traffic flow forecasts, an anticipatory signalisation is developed. Two routing protocols for urban road traffic guidance are transformed into time-dependent route guidance protocols including both current sensor values and forecasts for future points in time. A rule-based machine learning technique, a variant of an extended classifier system, is adapted to the problem of congestion detection, both for highways and for urban areas. Finally, approaches for a fully self-adaptive management process are outlined in which the system is enabled to react autonomously to congestion alarms.

Contents

Chapter 1

Introduction

1.1 Motivation

In many countries, the automotive industry contributes a big part to the overall economy, offering jobs for thousands. Novel trends, such as self-driving cars and clean and energy efficient vehicles play a leading role in the future of the world-wide car industry. Novel technologies work as incentives for the ongoing research in the automotive field, and its related areas. However, technological innovation and the increase in mobility also pose several problems. Especially in densely populated urban areas, traffic volume and traffic performance are rising. The car is still the number one means of transportation in 58% of all trips. This accounts especially for trips with distances up to 500 kilometres, where around 70% of all trips are made by car. For example, the average German citizen covers a distance of 39 kilometres per day [LNK+10]. The inefficient use of the existing infrastructure and the lack of space for new roads demands for new and improved solutions. Simply expanding the capacity of the road network is seldom an option due to limited transportation funds and a lack of public acceptance because of environmental impacts [Ber05].

The increase in mobility poses several challenges for future transportation systems. On the one hand, current research deals with new techniques related to mitigation of congestion, improvement of throughput or reduction of waiting times at red traffic lights. Traffic engineers agree that recurrent congestion due to demand exceeding capacity and poor signal timing account for about half of the total delay experienced by motorists, while non-recurrent congestion (due to road works, incidents, and weather) makes up the other half [Ber05]. This leads to a massive waste of time, fuel, and money [LNK+10]. On the other hand, negative environmental impacts due to greenhouse gas emissions by combustion motors promote global warming and climate change. As investigated by the German "Bundesministerium für Verkehr, Bau und Stadtentwicklung" in their latest report of 2008 [LNK+10], the CO_2 pollution is slowly declining since 2000, although the amount of 161 tons due to traffic in 2006 corresponds to a ratio of 18% of the overall CO_2 emissions in Germany. Regarding the European emissions, the transportation sector accounts for 25% of the total carbon dioxide emissions [DSD+12]. The decarbonisation of this sector can be promoted through greener, sustainable mobility, alternative energy sources, and more efficient traffic management processes.

The terms *smart mobility* and *sustainable mobility* stand for novel approaches to mitigate the negative impacts of road traffic by improving traffic flow, reducing the number of start-stop phases, and reducing the total travel distance. *Intelligent transportation systems* can improve the

efficiency of road networks by dynamically adjusting the signalisation to current traffic demands and by recommendation of shorter, faster, or more eco-friendly routes to motorists. Intersection management, routing, and congestion avoidance are key factors for improved mobility and better road network utilisation. Even if the primary objectives are not related to energy efficiency and reducing CO2 emissions, the implemented measures may still have an impact on them.

The dependencies between thousands of independent driving decisions and the information overload from numerous sensors throughout the traffic network make the decision-making too complex, even for traffic experts [MKM+16]. In general, it is impossible to anticipate all possible situations. The decision finding happens in a highly dynamic environment that changes with the execution of actions, and over time. Therefore, instead of preplanned solutions with fixed situation-reaction mappings, the trend goes towards adaptive systems, shifting the action-selection-process from design time to runtime. This lead to the emerge of several autonomic traffic control systems which autonomously adapt the control strategies to the monitored traffic conditions. However, the deployed traffic management systems are mostly only adjusting the signal plans to the recent traffic conditions within limited boundaries, whereas the actual congestion management is still executed by traffic engineers. Furthermore, the reactions to the monitored situations might be already outdated when the system adjusts itself. Therefore, novel approaches are needed that convert the existing reactive systems with pre-planned solutions to self-organised and anticipatory traffic management systems.

This thesis contributes several new approaches to the field of autonomic traffic management. In the following, concepts from the time series forecasting domain and modern machine learning approaches are applied to self-optimising intersection management, autonomous congestion detection, anticipatory route guidance, and road traffic control. These methods contribute to the overall goal to make the existing traffic control concepts more intelligent and more resilient against disturbances. Furthermore, the traffic network is enabled to provide fast, safe, and efficient transportation while minimising negative impacts on the environment.

1.2 Problem statement & objectives

> ”Prediction is very difficult, especially if it’s about the future.”
>
> Niels Bohr

Traffic forecasting systems have the potential to improve traffic conditions and to reduce travel delays by improving the utilisation of the available capacity. A traffic forecasting system utilises models to analyse real-time data from different sources to model and predict future traffic conditions. This problem is tackled by applying advanced forecast methods. In case of traffic management, the forecasting of traffic flow, velocity or travel time is necessary to optimise certain management strategies. In addition, anticipatory traffic management strategies are implemented to meet various traffic control, management, and operation objectives.

Organic Traffic Control (OTC) [PTB+11] resembles a robust and flexible traffic management system for urban traffic networks, founded on the principles of Organic Computing [MSvdMW04].

It autonomously adjusts the signalisation according to the recently monitored traffic measurements by extending standard fixed-time controllers. However, it only reacts to previously monitored sensor values. As *Prothmann* [Pro11] pointed out, predicting future traffic developments can be beneficial for the system's performance. To achieve this goal, the observer/controller architecture after which OTC is modelled will be extended in this work by a forecast module. Thereby, other organic computing systems can also easily integrate this new component in the future.

In a first step, instead of just reacting to events that have already occurred, the OTC system is extended to predict events in the near future. In a second step, the system is enabled to take countermeasures to minimise the negative effects of the predicted event. In general, the process works as depicted in Figure 1.1. First, the monitored sensor data is preprocessed (we usually have to deal with noisy and faulty detector data). The adjusted data is analysed and used to make short-term forecasts. Based on these insights, the system adjusts its behaviour and the next cycle starts. A more detailed introduction to OTC is given in section 2.2.

Figure 1.1: Process of a self-adapting system using forecasts.

Transportation agencies use long term forecasts to plan and implement policies and investment programs to accomplish objectives related to mobility. Still, the actual decision making and the deployment of the strategies is done by humans and not by the system itself. However, both traffic engineers and traffic participants must be aware of the considerable uncertainties that surround travel forecasts. This contributes to "the fact that forecasts are not perceived as being sufficiently accurate to be a decisive element of the decision-making" [Woo13]. This thesis targets at improving the short-term forecast accuracy of traffic flow with modern machine learning techniques. Furthermore, these forecasts are not only raised but the self-adaptive OTC system is enabled to autonomously utilise those forecasts for improving the route recommendation system and the optimisation process of the signalisation. Even if the forecasts are rather accurate, they still have a certain deviation from the actual observations. Therefore, the presented algorithms and methods incorporate forecasts with respect to their recent precision.

In urban areas, traffic flow is fluctuating more heavily than on highways as incoming streets and the stop-and-go generated by traffic lights influence the traffic flow patterns. This makes forecasting of these patterns more challenging compared to the more homogeneous flow on highways. By extending OTC with a module for time series forecasting, the problem of forecasting traffic flow in urban areas is addressed.

In conclusion, the main objectives of this thesis are:

- Extending the architecture for organic computing systems with a forecast module for time series.
- Utilising machine learning techniques to improve the accuracy of short-term traffic flow forecasts.
- Identifying beneficial ways to incorporate forecasts into a self-adaptive, decentralised traffic management system, e.g. OTC.
- Utilising forecasts within autonomous strategies with respect to the forecast's accuracies, e.g. distributed route guidance and autonomous adaptation of signalisation.
- Application of machine learning approaches to traffic related problems, such as congestion detection on highways and urban streets.

By implementing these objectives, it is expected that the following goals are achieved:

- Traffic safety is increased.
- The efficiency of the road network is increased.
- Environmental pollution is reduced.

The second and third point can be proven through realistic (macroscopic) simulations of real-world traffic networks. However, the first point is difficult to show, as it concerns complex inter-vehicle relations, located on the microscopic level. Therefore, we can only assume that traffic safety will increase, for example by reducing the number of congestion the frequency of crashes is likely to be reduced as well [MW10].

1.3 Contributions

This thesis contributes to the study and application of strategies for time series forecasting and methodologies from the machine learning domain to an equally considerable large domain, namely traffic engineering. Furthermore, the architecture for organic computing systems is extended by a forecast module, enabling these technical systems to become proactive. The remainder of this thesis is structured as follows.

The first part includes three chapters which give an overview of the state-of-the-art of traffic control, of time series forecasting, and of learning classifier systems.

Chapter 2 gives an introduction to traffic control and presents the self-organised traffic control system Organic Traffic Control (OTC) for urban road networks. Additionally, the observer/controller architecture from the organic computing domain, on which OTC is based, is introduced. The term "resilience" is discussed within the context of technical systems. Based on this definition the term "resilient traffic management" is defined. At last, a road map – which is partly based on [STH13a, STH16b, ST16a] – is outlined which presents ways to transform a traffic-responsive control systems into a resilient, anticipatory traffic management system. Thereby, the design pattern for organic computing systems is extended by a forecast module

for time series. The points of this road map also resemble the contributions presented in the remainder of this thesis.

The second chapter of this part (Chapter 3) gives an overview of related work on the forecasting of time series. Chapter 4 introduces the reader to the fundamentals of learning classifier systems. We will apply these systems to important issues concerning traffic management and forecasting, e.g. congestion detection and combination of techniques for forecasting of time series.

The second part of this thesis presents the contributions to the previously introduced state-of-the-art.

Chapter 5 – which is partly based on [STHA15, SSH16a, SSH16b] – shows an approach to apply a genetic, rule-based machine learning technique to time series forecasting. First, the general problem of time series forecasting is formalised and transformed into a machine learning problem. With the help of a machine learning technique, the forecast accuracy of an ensemble of different forecast techniques is improved. We apply the extended classifier system for function approximation (XCSF) to this task. Afterwards, we transfer this general approach to forecasting of traffic flow.

The second major contribution is a forecast-augmented adaptation of traffic signalisation. Chapter 6 presents how the concept of traffic flow forecasting can be used within OTC in a beneficial way. This chapter is partly based on [STH14, SH16, ST16b] and shows how the standard OTC is adapted to autonomously find the optimal signalisation not only based on historic data but also incorporating traffic forecast of the near future.
A study by the transportation research board [SU10] summarises the most widely deployed traffic control systems. However, these systems solely adjust signal timings in response to sensor measurements. Some systems use the terms 'forecast' or 'prediction', even so they do not utilise sophisticated forecast methods. They estimate traffic parameters, such as arriving of public vehicles, based on simple heuristics.
In this work, a novel concept for the adaptation of signal timings based on forecasts of traffic flow is proposed. Based on short-term forecasts of traffic flow, the distributed, self-adaptive optimisation of signalisation is executed. A mechanism for the anticipatory and traffic-responsive optimisation of signalised intersections in urban road networks is presented. A simulation study based on a real-world isolated intersection and an artificial road network shows that this approach has the potential to decrease travel times and pollution emission.

Third, two well-known Internet-based routing algorithms are extended for anticipatory routing in urban road networks. In chapter 7, a predictive route guidance system for urban networks considering short-term forecasts of traffic flow is implemented. Being partly based on [STH15a, STH15b, STH16a], the chapter presents how two existing non-predictive Internet-based shortest-path algorithms [PTL+12] can be extended with forecasts. Until now, these protocols solely approximate the shortest paths based on previous sensor measurements. However, as stated by [WKS00], predictive route guidance outperforms non-predictive approaches. Consequently, we propose a novel approach, estimating the shortest routes within a road network using both historic sensor values and forecasts of future developments. The adapted variants depict the time-varying conditions of traffic by making forecast of future travel flows. Furthermore, the Dijkstra algorithm is adapted to calculate the shortest paths through the road network

based on the time-dependent traffic flow representations and with respect to the estimated forecast accuracies. The evaluation with artificial networks showed that an additional positive effect is the reduction of pollutant emission (eco-aware routing).

Fully resilient traffic management should also cover the possibility to react to disturbances besides the "normal" capacity-related problems. Therefore, anomalies in terms of incidents have to be considered. This consists of two basic parts: a) detecting these incidents automatically and b) including the knowledge about occurred incidents into the traffic management system's decision process.

The fourth contribution covers the first of these two points. Chapter 8, which is partly based on [SH17b], provides an overview of congestion detection algorithms. A genetic, rule-based algorithm, a learning classifier system [BK05], is adapted to solve this task. The benefit of these techniques is shown with data sets of real-world measurements from inductive loop detectors. Current concepts in literature try to detect such events with different traffic-pattern-based algorithms, but often focus on highways only (e.g. the California algorithm [PK76]). The challenge is to transfer these highway-based concepts to urban areas, where traffic patterns caused by red traffic lights can be similar to those caused by incidents.

The fifth contribution deals with this issue. The previously adapted learning classifier system, the XCSR, is further adapted and implemented to be used for congestion detection in metropolitan areas within the distributed architecture of OTC. Afterwards, ways to include the congestion notifications (or alarms) into a fully self-organised traffic management system's decision process are presented. Being partly based on [SH17a], chapter 9 discusses approaches how these congestion alarms can be used within a self-adaptive traffic control system, such as OTC. Furthermore, XCSR is compared with another well-established algorithm for congestion detection with simulations of real-world scenarios.

All these concepts have the potential to increase resilience in the presence of uncertainty, and to improve levels of service. It is expected that a better utilisation of the network capacities and a reduction of congestion can be achieved by interactive traffic management, harmonising the traffic flow and by consistent guidance of traffic streams. In the long term, urban road networks become more robust, eco-friendly, and reliable.

The concluding chapter 10, gives an overview of the presented work and outlines promising future research opportunities.

1.4 List of publications

The work and results presented in this thesis have been published in the following publications:

Book chapters

- *Matthias Sommer*, Sven Tomforde, and Jörg Hähner: "An Organic Computing Approach to Resilient Traffic Management". In *Autonomic Road Transport Support Systems*, pp. 113-130, ISBN 978-3-319-25808-9, 2016.

Articles

- *Matthias Sommer*, Sven Tomforde, and Jörg Hähner: "Using a Neural Network for Forecasting in an Organic Traffic Control Management System". In *Proccedings of the International Workshop on Embedded Self-Organizing Systems (ESOS)*, pp. 1-6, USENIX, 2013.
- *Matthias Sommer*, Sven Tomforde, and Jörg Hähner: "Resilient Traffic Management with Organic Computing Techniques". In *Proceedings of the 1. International Systems Competition on Autonomic Features and Technologies for Road Traffic Flow Modeling and Control Systems, held together with the 16th International IEEE Conference on Intelligent Transport Systems (IEEE-ITS)*, pp. 1-8, IEEE, 2013. (**Winner of the ARTS Competition at IEEE-ITS13**)
- *Matthias Sommer*, Sven Tomforde, and Jörg Hähner: "Learning to Predict: Automated Management and Correction of Prediction Techniques for Traffic Flows within a Self-organised Traffic Control System". In *Proceedings of the 11th International Congress on Advances in Civil Engineering (ACE)*, pp. 1-6, 2014.
- *Matthias Sommer*, Sven Tomforde, and Jörg Hähner: "A Systematic Study on Forecasting of Traffic Flows with Artificial Neural Networks". In *Proceedings of the 28th International Conference on Architecture of Computing Systems (ARCS)*, pp. 1-8, IEEE, 2015.
- *Matthias Sommer*, Sven Tomforde, and Jörg Hähner: "Forecast-based Route Recommendations in Organic Traffic Control". In *Proceedings of the 28th GI/ITG International Conference on Architecture of Computing Systems - ARCS Workshops*, pp. 11-12, 2015. (**ARCS 2015 Best Poster Award**)
- *Matthias Sommer*, Sven Tomforde, and Jörg Hähner: "Demo: Proactive Re-Routing of Urban Traffic based on Traffic Flow Forecasts". In *Proceedings of the 2nd International Systems Competition on Autonomic Features and Technologies for Road Traffic Flow Modeling and Control Systems*, pp. 1-8, 2015. (**3rd place at the 2nd ARTS Competition**)
- *Matthias Sommer*, Sven Tomforde, and Jörg Hähner: "Learning a Dynamic Re-combination Strategy of Forecast Techniques at Runtime". In *Proceedings of IEEE International Conference on Autonomic Computing (ICAC)*, pp. 261-266, IEEE, 2015.
- *Matthias Sommer* and Jörg Hähner: "Anticipatory adaptation of signalisation based on traffic flow forecasts within a self-organised traffic control system". In *Proceedings of the 5th International Conference on Transportation and Traffic Engineering (ICTTE)*, pp. 1-6, MATEC, 2016.
- *Matthias Sommer*, Anthony Stein, and Jörg Hähner: "Ensemble Time Series Forecasting with XCSF". In *Proceedings of the 10th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pp. 150-151, IEEE, 2016.
- *Matthias Sommer*, Anthony Stein, and Jörg Hähner: "Local Ensemble Weighting in the Context of Time Series Forecasting Using XCSF". In Proceedings of the IEEE Symposium on Computational Intelligence and Ensemble Learning (CIEL), pp. 1-8, IEEE, 2016.
- *Matthias Sommer*, Sven Tomforde, and Jörg Hähner: "Forecast-augmented Route Guidance in Urban Traffic Networks based on Infrastructure Observations". In *Proceedings of the International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS)*, pp. 177-186, SCITEPRESS, 2016.

- *Matthias Sommer* and Jörg Hähner: "Learning Classifier Systems for Road Traffic Congestion Detection". In *Proceedings of the 3rd International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS)*, pp. 142-150, SCITEPRESS, 2017.
- *Matthias Sommer* and Jörg Hähner: "Adapting Signal Timings to Automated Incident Alarms within a Self-organised Traffic Control System". In *Proceedings of the 3rd International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS)*, pp. 203-210, SCITEPRESS, 2017.

Technical reports

- *Matthias Sommer*, and Sven Tomforde: "Concepts for Resilient Traffic Management based on Organic Computing", Technical Report at the University of Augsburg, Faculty of Computer Science, pp. 1-28, 2016.
- *Matthias Sommer*, and Sven Tomforde: "Dynamic ensemble forecasting of traffic flow by means of machine learning techniques". Technical Report at the University of Augsburg, Faculty of Computer Science, pp. 1-26, 2016.

Part I

Overview

Chapter 2

Resilient traffic management

The vehicular traffic domain is a vivid research field, both for industrial and academic research institutions. On the one hand, emerging technologies, such as autonomous, self-driving cars [Bir14, Int14], car-to-car communication [BBD+08], and traffic-adaptive control systems [SS10, PTB+11], have the overall goal to optimise the existing traffic infrastructure towards a more efficient and sustainable utilisation of the road network. On the other hand, negative impacts on the environment due to the increase of mobility have to be faced, especially in urban areas. Consequently, this leads to an increase in pollution, a raising number of incidents, and an inefficient use of the transportation system [Pat94].

Most of the currently installed adaptive traffic management systems (such as SCOOT [RB91] and COMPASS [MW91]) rely on centralised control centres, where all data is gathered and processed, and decisions about necessary adaptations of the underlying control strategy are made. This results in a single point of failure and a high demand of computational power. Centralised solutions are not able to cope with expanding networks and future demands. Urban road networks are characterised by a great number of signalised intersections in vicinity that need to be monitored and optimised. Due to the complexity of the underlying mathematical network flow model, even the online optimisation of a single signalised intersection is hardly feasible with analytical methods [KLS02]. Above all, the dynamic characteristics of traffic, the unpredictable behaviour of humans, and the highly complex dependencies between several streams throughout the road network make it a challenging field for self-adapting and self-organising solutions. Consequently, novel design paradigms lead to decentralised, autonomous, and self-organised traffic control systems, resulting in integrated transportation systems, such as InSync [SS10], and research projects, such as Organic Traffic Control (OTC) [PTB+11].

Usually, the deployed controller-based control systems rely on fixed-time signalisation that was optimised manually by traffic engineers during design-time. Obviously, these static signal plans can not be optimal in every situation and are prone to obsolescence due to changing traffic demands [BB86]. In contrast, a distributed, autonomous, adaptive traffic control system (ATCS), such as OTC, is able to select the most appropriate phase durations, green times, and cycle times at each intersection based on the current traffic conditions. OTC learns the impact of this situation-dependent selection in order to improve its behaviour over time while respecting safety corridors.

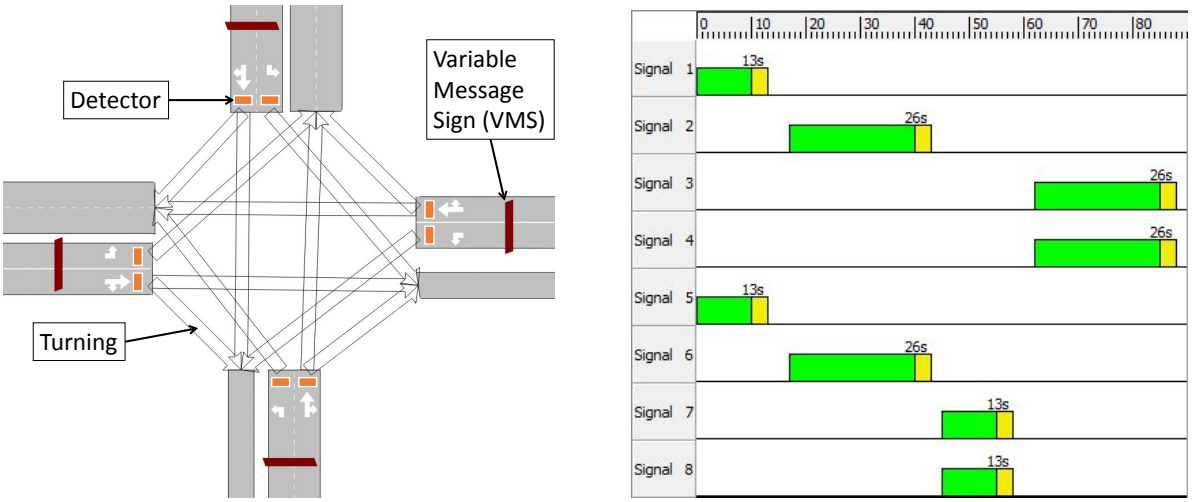
Most importantly, classic, responsive systems are not able to make adjustments fast enough to prevent disturbances. The monitored situations are often outdated when the adaptation takes place. Therefore, the system needs to be equipped with mechanisms that allow preventative measures. Traffic forecasting is a key enabler for successful control strategies – a variety of

different techniques can be found in literature [BF12]. The study of forecasting traffic flow has attracted considerable attention in recent years. However, most prior research has been limited to the investigation of forecasting methods. So far, little effort has been devoted to the integration of these forecasts into real-world traffic management systems.

The remainder of this chapter is structured as follows. Initially, a brief introduction of terms and definitions from the traffic control domain is given. Moving on, OTC, a self-organising traffic control system for urban road networks, is introduced. Afterwards, the term *resilience* is defined and several definitions are given. Finally, new approaches to improve OTC and to make it more resilient by using short-term forecasts of the future traffic flow are presented. Making forecasts of future system states can make complex technical systems more robust against failures. A new module for the creation of forecasts at runtime is presented as well as how these forecasts can be integrated beneficially in OTC. A discussion on how this can lead to higher resilience concludes this chapter.

2.1 Terms and foundations

This section briefly introduces the corresponding state of the art in traffic engineering. First, some basics about traffic engineering are explained by means of a conceptual traffic intersection. The four-armed intersection shown in Fig. 2.1a consists of four approaching and four leaving sections. The intersection's topology is defined by the turnings between these sections. Each turning might be signalised by its own traffic light or it might share one traffic light (such as turnings for straight-ahead and turn-right). It is assumed that detectors are installed at each turning, typically realised as induction-loops in the street surface. Additional variable message signs (VMS) can give drivers information about the current state of traffic or additional route advice.



(a) Exemplary four-armed intersection with twelve turnings.

(b) Signal plan with eight signals and a cycle time of 90 seconds.

Figure 2.1: An exemplary four-armed intersection and the corresponding signal plan.

2.1.1 Terminology

Traffic lights allocate green, yellow and red times to the various turning movements of a signalised intersection. These turnings that direct traffic through the intersection without a potential conflict form so-called *signal groups* (see Fig. 2.2). A *signal group* consists of one or more traffic lights that can not be switched independently, therefore displaying the same signalisation at each point in time for different non-conflicting traffic streams. Several signal groups commonly showing green light at the same time form a *phase*. *Phase transitions* control the switching between different signal groups. They assure sufficient *clearing times* between the end of the previous and the start of the next turning movements. A *signal plan* is made up of the phase sequence, the phase transitions, and the length of each phase for all signals at an intersection. It visualises the green, yellow, and red times for each of the intersection's signal groups. As an example, consider the signal plan in Fig. 2.1b that shows the duration of green and yellow periods for signal groups A to D from Fig. 2.2. At other times, the so-called *interphases*, the traffic lights are flashing red in order to avoid accidents with conflicting streams. The time period in seconds for one complete iteration of the signal plan is called *cycle time*. After completing a full *cycle* (one complete sequence of signalisation), the currently executed signal plan is repeated from the beginning.

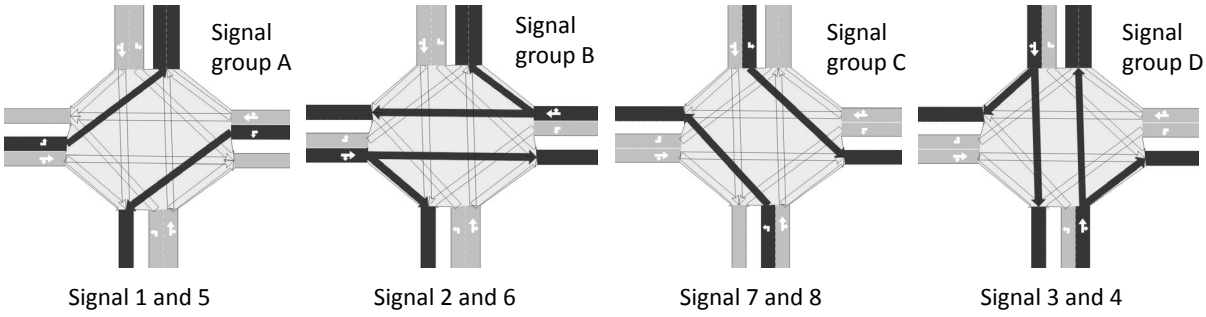


Figure 2.2: Setup of an exemplary signal plan for the intersection shown in figure 2.1.

Typically, traffic engineers try to design the best possible signal plans for historic records of traffic movements. Different metrics help to quantify the performance of signalisation strategies. For instance, in order to reduce the waiting times due to red lights, one might utilise the *level of service* (LOS). This qualitative measure is defined as the sum of the flow-weighted waiting times for each turning $t \in T$ of the respective signalised intersection:

$$LOS = \frac{\sum^{t \in T} delay * flow}{\sum^{t \in T} flow} \tag{2.1}$$

The result is discretised into quality levels from A (free flow) to F (congestion). Other network-wide goals include the minimisation of the total number of stops or the minimisation of the overall fuel consumption. In most cases, isolated intersections are designed based on the LOS metric. The increase in green phase durations tends to increase the intersection's throughput (or capacity). However, it also increases the waiting times for other traffic participants. Accordingly, decreasing green durations decreases waiting times but also reduces the intersection's capacity.

2.1.2 Traffic control

Deriving the best trade-off between both aspects (low delay and high throughput) resulted in the establishment of urban traffic light control systems as a prominent research domain in academia for decades. Additionally, several industrial systems are available, since traffic control has a large environmental and economical impact. In general, two approaches are distinguished: fixed-time control and traffic-actuated control. Both aim at optimising the traffic network flow by adjusting the signalisation to meet the estimated traffic demands. In case of fixed-time signalisation, this optimisation is usually done by a traffic expert before the deployment. In case of adaptive systems, the optimisation is performed at runtime through heuristics, search techniques, or a combination of heuristics and traffic engineering concepts [SU10]. Variables that can be optimised are cycle lengths, green time durations, split offsets, and phase sequences.

Further information on the terminology, setup, operation, and maintenance of fixed-time and adaptive traffic light controls are available in the *NEMA Standards Publication* [Nat03] (for traffic control in the U.S.), and in the *Richtlinien für Lichtsignalanlagen* (RILSA [fSuVAVuV10]) for traffic control systems in Germany. The following section briefly introduces the basics of fixed-time and adaptive traffic control.

Fixed-time control

The first category of traffic control systems uses fixed-time based signalisation with a limited set of predefined signal plans for each signalised intersection. Fixed-time control relies on switching pre-defined phases with static phase durations and ordering. These signal plans are developed by traffic engineers at design time based on their domain knowledge and previously recorded, historical data. A fixed sequence of phases with predefined durations is repeated over and over. Typically, a day-time dependent switching of signal plans takes place, to account for different traffic demands during day and night time and peak hours. As a consequence, these signal plans are not responsive to dynamic and changing traffic demands, and tend to become outdated over time [BB86]. However, due to their simplicity and low cost, they are still in use today.

Adaptive traffic control

Traffic-actuated control, in contrast, allows skipping phases when no vehicles are waiting, and for increasing the green times within certain boundaries. Adaptive traffic control systems (ATCS) are being used since the early 1980s. They adjust signal timings in real time, based on the momentarily monitored traffic conditions, demands, and system capacity [SU10]. They require traffic detectors to estimate the current traffic demands, communicational infrastructure between the local controllers, and/or a central controller for collaboration and monitoring. Just to name a few ATCS's that are deployed in the field: SCOOT [RB91], SCATS [SD80], OPAC [GPA02], and ACS-Lite [SBS$^+$08] (ordered by year of launch). ACTSs usually include algorithms that adapt green times, cycle time, offset, phase length, and/or phase sequence to optimise the traffic throughput at signalised intersections. Certifications, e.g. according to the NEMA standard (common for the US, [Nat03]) or the VS-Plus standard [Swi08], ensure the required degree of reliability of traffic-response controllers.

2.2 Organic Traffic Control

The complexity of modern technical systems increases as they are interconnected with other systems and devices. This makes them more and more complex to design and to maintain for administrators. Traditional system design approaches tend to fail in addressing this problem. *Organic Computing* (OC) [PTB+11] proposes a new design-paradigm to cope with these challenges by reducing the complexity of these systems. The OC initiative [MSvdMW04] postulates to master complexity in technical systems by equipping them with characteristics of natural – or *organic* – systems. On the one hand, this means to enable technical systems with capabilities of self-adaptation and self-optimisation of their runtime behaviour, and consequently with a robust self-organisation mechanism. However, IT-experts are still able to control certain subcomponents of the system. On the other hand, this results in moving traditional design time decisions to runtime, and into the responsibility of the systems themselves. Among others, current OC systems include autonomous traffic light optimisation [PTB+11, STH16b], Cloud Computing applications [SKTH16], and organic production cell [Tom11]. Due to the dynamic and highly complex nature of vehicular traffic, the control of signalised intersections is an ideal test bed of OC principles – which resulted in the development of the *Organic Traffic Control* system (OTC) [Pro11]. OTC extends existing fixed-time controllers, equipping them with OC capabilities. OTC continuously adapts to the behaviour of traffic participants.

The OTC system consists of three major modules: 1) Autonomous and self-optimising control of traffic signals at intersections, 2) distributed coordination of neighbouring intersection controllers to establish progressive signal systems and 3) infrastructure-based route recommendations with Internet-inspired routing protocols.

2.2.1 Observer / controller design pattern

Figure 2.3 depicts the single-layered observer/controller architecture known from Organic Computing [PTB+11]. This framework aims at enabling systems to automatically adapt to changing environments, to learn the best adaptation strategy, and to explore new behaviour [BHT12]. It is often used as a reference design model for organic systems [TSHMS09, Tom11, Tom12]. The underlying system under observation and control (SuOC) is monitored by the observer and actions are applied by the controller. In the following, a brief introduction to the various sub-parts of this architecture is given: the observer, the controller, and the SuOC.

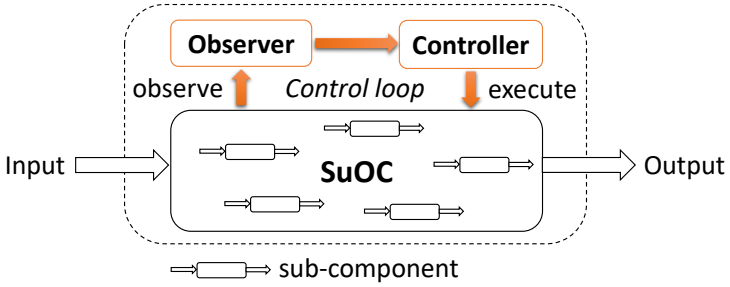


Figure 2.3: Generic observer/controller architecture, monitoring and controlling a system under observation and control (SuOC). For a detailed view of the observer see figure 2.4.

System under observation and control

The SuOC resembles the underlying productive system. It is assumed that it contains parameters that can be adapted at runtime and that it offers possibilities to continuously measure its performance. The SuOC is controlled by the O/C architecture. However, it is still able to provide its basic functionality without control by the observer and controller and will provide its services even if the upper layer fails. Usually, the SuOC has several entities, sensors, and actuators that enable the regulation of its behaviour. The O/C architecture makes use of these interfaces to regulate and optimise the behaviour of the SuOC at runtime. In the context of this thesis, the system is represented by a parametrisable traffic light controller which offers interfaces for data retrieval and the application of actions.

Observer: Parameter selection and forecasting

The observer's task is to monitor the underlying SuOC, and to provide a description of its current state. Figure 2.4 shows a detailed view of the observer which was extended by a time series forecasting module to reach the goal of this thesis.

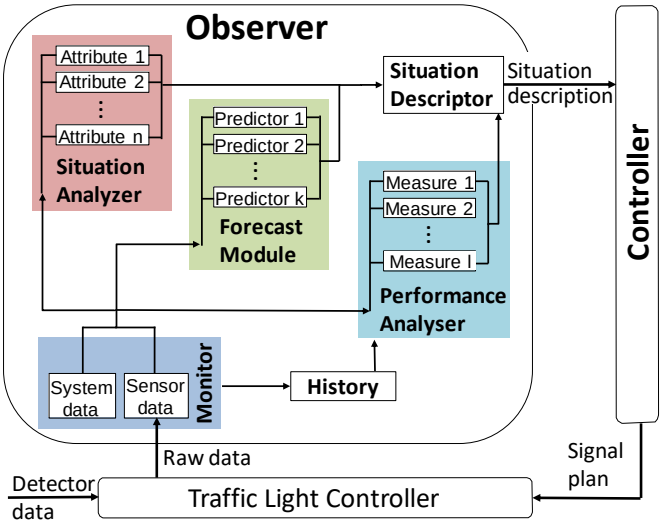


Figure 2.4: Detailed view of the observer at layer 1. To add resilience to organic systems, the standard observer architecture is extended in this work by a forecast module.

First, the *Monitor* receives raw data recorded by sensors. This raw data is often noisy and might be subject to disturbances (for example due to malfunctioning sensors). Therefore, the raw data is preprocessed. For example, it can be filtered or smoothed. The history of previous data recordings is stored for later evaluation. Afterwards, the processed data is passed on to other components, such as the *Situation Analyser*, the newly introduced *Forecast Module*, and the *Performance Analyser*. Second, the *Situation Analyser* builds a situation description of the current system conditions. The measurements from the sensors are converted into a rolling detection interval. Third, the *Performance Analyser* retrieves statistical information about the

system's performance derived from the current monitored data, and the history of previous measurements. Fourth, the *Forecast Module* generates forecasts based on historical and recent data for future points in time. These forecasts and the performance measures can be used as additional input for the situation description. In section 2.4.1, this novel component is explained in more detail. Fifth, the *Situation Descriptor* receives the current situation description, the forecasts for future time steps, and the performance measures, and creates one overall situation description. Finally, this description is passed on to the *controller*, where it serves as basis for the following action-selection process.

Controller: Online adaptation

The controller selects suitable actions to be executed on the SuOC based on the situation description given by the observer and with respect to the system's goals. This action-selection process is often done with the help of machine learning techniques. The benefit of such machine learning algorithms is that they are able to improve their knowledge at runtime and to provide better decisions over time. The controller follows a safety-based approach, so that the parameter changes on the SuOC are only within predefined boundaries.

2.2.2 Multi-layered observer/controller architecture

Based on this generic design paradigm, the self-organised traffic control system OTC was implemented. Figure 2.5 illustrates the architecture of the OTC system. Autonomous and self-optimising control of traffic signals at intersections is achieved by applying this architecture to the control of traffic light controllers (TLC) at urban intersections. OTC's goal is to monitor, process, and disseminate information about the current conditions of the road network. The system under observation and control (SuOC) at layer 0 is represented by an industry-standard, parametrisable TLC which is extended by the OTC logic. The TLC can work on fixed-time signal plans or can be implemented according to traffic-responsive solutions (such as NEMA [Nat03]). The TLC is responsible for adapting phase durations (i.e. green times of traffic lights) according to the currently observed traffic conditions at the particular intersection. Thereby, a safety-oriented, self-learning concept is applied to allow for a continuous self-optimisation process while simultaneously staying within controllable boundaries of system behaviour. OTC works in a fully decentralised mode – meaning that one individual instance of OTC deployed on a TLC controls one signalised intersection only. This optimisation is done in a two-step fashion: online parameter selection at layer 1 and gaining of new knowledge offline at layer 2.

Layer 0: System under observation and control

In the context of OTC, the SuOC situated at layer 0 is an industry-standard, parametrisable TLC. It provides input and output interfaces for OTC to receive data and to actuate new signal plans. Physical sensors monitor traffic streams (e.g. via loop detectors, license plate matching, or equipped vehicles) and forward the sensory data to layer 1. The controller uses actuators for the physical execution of new signal plans and adapted green times at the SuOC level. Figure 2.5 illustrates the schematic setup.

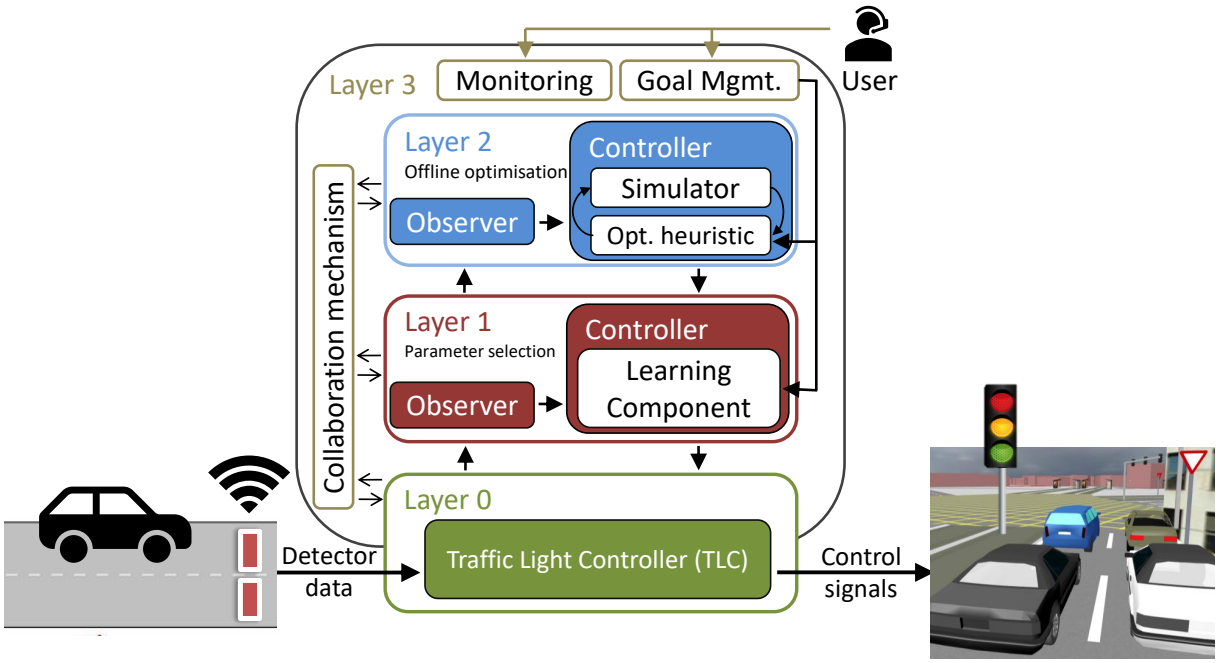


Figure 2.5: Self-organised adaptation process of signalisation by extension of a parametrisable FTC with OTC.

Layer 1: Online learning

Layer 1 is responsible for the online learning process. The observer aggregates and analyses the current traffic situation and provides a situation description of the current traffic conditions. This input is used as decision basis for the signal plan selection by the controller. In OTC, the controller is realised as a rule-based learning classifier system. The controller performs two tasks: 1) the success of previous decisions is estimated (in terms of averaged delays in front of red lights), and the corresponding rules are updated; 2) the most promising rule matching the current traffic condition is selected – resulting in a physical modification of the green times of the TLC at layer 0. Layer 1 operates on existing rules only and is restricted to exclusively using similar rules. In other words, the situation the rule has been designed for, must be highly similar to the currently observed one. In case that no matching rule was found, the offline optimisation process at layer 2 is activated. For safety reasons, only valid rules that guarantee a secure traffic flow will be generated.

A modified variant of Wilson's machine learning technique, the *extended classifier system* (XCS) [Wil95] (see chapter 4) has been integrated into the controller. In previous work [Pro11], the standard XCS was adapted to learn the best mapping of traffic situations to green phase durations. This XCS variant tries to evolve rules that encode the best signalisation for traffic conditions monitored at the local intersection. In conclusion, the current traffic condition at the local intersection serves as input to the classifier system.

Table 2.1 illustrates an example of an XCS' rule set. The condition part of each classifier defines in which situation interval the rule is applicable. It is defined as a vector containing the moni-

Table 2.1: Exemplary excerpt of a classifier set of XCS within OTC.

ID	Condition: Situation (veh/h)	Action: TLC	P	ϵ	ϕ
1	[{44, 65}, {132, 150}, ...]	TLC_1	48	.02	97
2	[{87, 103}, {15, 27}, ...]	TLC_2	39	.03	73
	.				
	.				
	.				

tored traffic flow for all turning movements of the intersection. The same situation description also serves as input to configure the simulator at layer 2 when evolving a new rule for a currently unknown traffic situation. The TLC settings are stored as action part – meaning the green durations for all phases of the intersection which are executed in case the rule is chosen. The parameters P, ϵ, and ϕ resemble the prediction value, the prediction error, and the fitness of each rule, representing its expected reward, its accuracy, and its quality. These parameters are reinforced over time, based on the benefit of the according rule on the environment. The benefit (or reward in terms of XCS) of each applied TLC is estimated based on the average delay times occurring at the intersection. The reward is subject to a minimisation process. Equation 2.2 defines the reward function:

$$Reward(x) = \frac{\sum^{t \in T} f_t \cdot d_t}{\sum^{t \in T} f_t} \tag{2.2}$$

where x is the particular intersection, and T is the set of the intersection's turnings. The variables f_t and d_t denote the flow (in vehicles per hour), and the average delay (in seconds) for a turning $t \in T$. The goal is to minimise the reward which is the same as reducing the average delay for incoming roads with heavy traffic flow.

Further details on the learning mechanism and the reinforcement process can be found in [PRT+08]. A more detailed introduction to XCS is given by [BW03].

Layer 2: Offline optimisation

In order to be able to react appropriately in case of unknown situations, layer 2 is responsible for generating novel knowledge within a sandbox environment. Due to safety reasons, novel rules are only created in simulations at layer 2 and checked for validity – this is also called "sandbox-learning". A simulator which is configured with the intersection's topology and the particular traffic condition is coupled with an optimisation heuristic and an evolutionary algorithm to find the best possible setup of green phase durations with regard to the received traffic situation. The simulation is executed by the traffic simulation tool Aimsun [BC02] and combined with a genetic algorithm. The genetic algorithm creates several valid signal plans with different green time durations. Its general scheme is as follows: The algorithm starts with a set of randomly generated initial solutions. Some of the existing high quality solutions (parent rule) are selected and new offspring rules are created by randomised mutation and crossover operations. The parents and the offspring form the set of solutions for the next iteration. This

process is repeated until a fixed number of iterations have passed or a solution has reached a certain quality. Finally, the signal plan with the lowest estimated average delays is returned to the controller at layer 1, where it is added to its knowledge base. The new rule can be applied as soon as a matching situation occurs. This single-objective optimisation only considers one criterion. In contrast, multi-objective approaches try to find the optimal solution considering several – possibly contradicting – objectives. Reducing delay times usually leads to shorter cycle times whereas the minimisation of stops increases the cycle length by extending the duration of green phases.

Layer 3: Monitoring and goal management

Layer 3 provides an interface to the users and administrators of the system. Administrators and traffic experts can define new system goals, and are able to monitor the current behaviour and performance of each controller of the system. Additionally, layer 3 offers mechanisms for communication and collaboration between neighbouring systems via defined interfaces. Thereby, connected systems are able to exchange data or to negotiate configuration settings.

2.2.3 Real-world deployment

For the deployment of OTC in a real-world environment, a number of technologies are necessary and certain preconditions have to be met. First, sensors have to monitor the current traffic conditions. This can be realised with inductive loop detectors which resemble a cheap and well established technique in traffic management applications [PX06]. Second, the OTC logic works as an extension of the existing logic of the TLCs deployed at signalised intersections. It is assumed that these TLCs are parametrisable and that more computational power is necessary for fast and reliable computation. Third, the dynamic route guidance needs an interface, such as variable message signs, to visualise the route recommendations to the road users. These message signs can also be used by the incident detection component for indicating disturbed sections. Fourth, communication technology for the exchange of data between adjacent intersection controllers must be available. Besides these characteristics, OTC does not need further changes – especially no sophisticated detection and analysis devices. At last, the controllers have to be formally verified and certified by the responsible authorities. However, a formal verification of OTC lies beyond the scope of this thesis.

2.3 Introducing resilience into Organic Traffic Control

OTC and current traffic-responsive control systems react to the recently observed conditions – typically measured in terms of traffic flow. This approach has several drawbacks. For example, measuring traffic flow is always a trade-off between stable and recent trends (statistics taking intervals of varying length into account). More importantly, the observed flow values describe the past conditions rather than those upcoming. Independent of the actual traffic control strategy, the prediction of traffic volumes is an active field of research. Current approaches rely on means of statistical analysis to detect and extrapolate trends or they make use of averaged daily load curves. Daily load curves assume standard behaviour for classes of weekdays (for example

work days, week ends, holidays). In this work, *resilient traffic management* is introduced to add robustness to OTC. This goal can be reached with features, such as improved signalisation with forecasts, proactive rule generation, anticipatory route guidance, and automatic congestion detection. These objectives will be covered in the remainder of this thesis.

2.3.1 Term definition: Resilience

The main purpose of adding adaptation and self-optimisation capabilities to technical systems is to create robust and flexible solutions. In the context of OTC, all efforts to optimise the existing infrastructure aim at the overall goal to make the road network more resilient against failures. Several authors have suggested different definitions for the term *resilience* in technical systems. Until now, there has been some disagreement about a common definition. In the following, several definitions are presented and their similarities and differences are outlined.

A number of researchers have argued that resilience is not only to be seen as **reactive ability** (for example reacting to environmental changes with corresponding actions) [LH05]. In a recent study, *Välikangas* [VÏ10] stated that resilience incorporates both the **proactive capacity** to take counter-measures before its final necessity, and the *reactive capacity* to recover from degrading states after disturbances. *Legnick-Hall* and *Beck* [LH05] identify the role of resilience as a **proactive strategy** building on “forecasting and pre-emptive adjustment[s]”, and a **reactive strategy** as the ability to “meet every environmental change with a corresponding [..] action”. In this sense, *Wieland* and *Wallenburg* [WW13] name the first strategy **robustness** and the second one **agility**. They define robustness as the ability to “withstanding risks", and agility as "rapid response to events”. They conclude that both strategies have a positive effect on a system’s performance. *Pimm* [Pim84] suggests to measure resilience as the time a process needs to return to equilibrium after a perturbation. *Sterbenz et al.* [SeH$^+$13] add **survivability** as another aspect of resilience. They define survivability as “the ability to tolerate the correlated failures that result from attacks and large-scale disasters”. *Hellmann et al.* [HSG$^+$15] apply the concept of survivability to measure the type of resilience. In the context of traffic control systems, *Immers et al.* [ISY04] define resilience as “the capability [..] to repeatedly recover, preferably within a short time period, from a temporary overload.”

In conclusion, for this work the term *resilient* is defined as “proactive robustness”. This means that the control mechanism encapsulating the organic capabilities (in terms of organic computing) of the system does not only react to detected disturbances and dissatisfying system performance but that it also uses techniques to predict future developments and detect upcoming problems. One technique that can be used, and which will play a crucial role in this thesis, is forecasting of time series. In case an undesired environmental state is predicted (i.e. disturbances or shortages characterise the predicted situation), the self-organised control mechanism will guide the system’s behaviour in such a way that these disturbances and shortages will be prevented from occurring. Resilience is therefore not only defined through *robustness* and *proactiveness*, but also incorporates *flexibility*, and *reliability* (see figure 2.6). In the following, each of these key components is discussed in more detail.

1) Robustness: Although a system can foresee certain disturbances that will have a negative affect, not all influences are predictable. According to [BCFM11], a system is robust if it “can

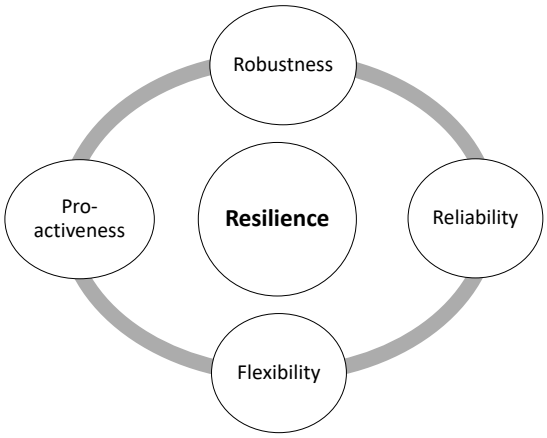


Figure 2.6: Key components of resilience.

still fail, but is less likely to transition to a partially-comprised state". *Master* [Mas06] defines robustness as the "ability to resist failure". In the context of Organic Computing, robustness is defined as "the capability to maintain a required behaviour or functionality in spite of a certain range of parameter variations" [MSSC+11]. This variations can be caused by internal changes within the system or by external influences from the environment [BHT12]. Meaning, a system is robust if the possibility of a system failure or of misbehaviours is low. The less a system is disturbed by malfunctions or by external influences, and the lower its recovery time is, the higher is its robustness (or fault tolerance). Still, a system is expected to recover from a disturbed state towards a state where it is at least able to provide essential functionality. In this context, *robust* means that the system is able to deal with a certain set of disturbances by keeping the system performance within a predefined corridor or at least guaranteeing to guide it back within a certain period of time.

2) Flexibility: In case of disturbances or new objectives, a system has to adapt to maintain its ability to fulfil its assigned task(s). During or after a disturbance, the system has to move back from a undesired condition to an acceptable or optimal state. This concept is similar to the definition of *agility* defined by [WW13]. But even in absence of negative impacts, the system might need to adapt to new goals defined by the user or react to changes in its environment to obtain its optimal performance. Hence, the sub-parts of a system can "modify the behaviour because of certain changes of their parameter values or of objectives" [MSSC+11] during runtime within predefined boundaries. However, the meaning of flexibility and the measurement is depended on the particular application domain [BHT12].

3) Proactiveness: As stated before, several researchers see proactive capabilities as one necessary characteristic for a resilient system. This includes two tasks: 1) making forecasts of possible future internal system states and of external influences, and 2) making proactive adjustments as to these predictions. Proactiveness comes with the cost of uncertainty and raises several questions for further research: How can forecast be derived in a precise fashion? How can forecasts be utilised to improve the system performance? How can the negative effects of the uncertainty deriving from the deviations between the predictions and the actual behaviour be mitigated?

Which forecast horizons have to be covered? These important questions will be subject for discussions in later chapters.

4) Reliability: In the context of road networks, Immers et al. [ISY04] define reliability as the "user-oriented quality of the transportation system". They argue that a robust system is necessary to offer "a high degree of reliability to the user". By this definition, a road network is reliable "if expected and actually experienced travel times closely agree". A greater number of unexpected delays makes the system unreliable.

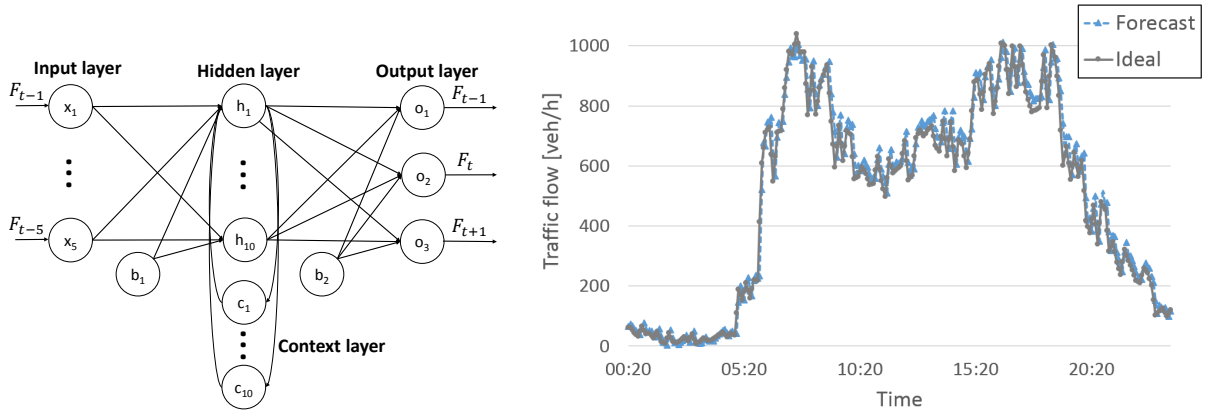
In terms of the previous definitions, the main goal of this work is to transform *robust* organic computing (OC) systems, especially OTC, into *resilient* ones by generating forecasts of future developments of the surrounding environment or of their internal behaviour. Until now, organic systems only had the ability to react to past events. In this thesis, their control mechanisms are improved by making use of forecasts. This is achieved by introducing a novel forecast module for time series within the observer component. Thereby, negative impacts can be anticipated and their influences can be reduced. Within this thesis, we refer to this property as *proactive robustness*.

2.3.2 Resilience through forecasting and machine learning

Short-term forecasting of traffic flow is an important aspect for intelligent traffic systems [ST16a]. A variety of forecast techniques has been developed to forecast traffic flow for future time horizons [STH13b, STH15a]. Typically, forecast techniques take the last recorded traffic flow into account to predict the estimated traffic conditions of the near future. There are also some techniques that rely on aggregated historical information (for example *daily load curves* [CKWS04]), or a combination of both. Furthermore, advanced forecast techniques consider additional input parameters, such as traffic density, current weather conditions, or data from adjacent roads (such as kalman filters [GLW97] or artificial neural networks (ANN) [DC97]). Figure 2.7 exemplarily shows that ANNs are capable of accurately forecasting traffic flow – for example that the deviations between the one-step forecasts and the actual values are low. An Elman recurrent network with one hidden layer and ten hidden neurons is used to forecast the traffic flow in five minutes using the last five minutes of recorded traffic flow as input [STH13b].

Other approaches focus on network-wide forecast models. They are neglected in this work, since forecasts are only considered within the control strategies of a single intersection. The focus lies on decentralised approaches for urban areas where traffic data can be obtained from sensors and (traffic-responsive) traffic lights. In order to allow OTC to proactively adapt to changing traffic demands, the system makes use of forecasts. Forecasts of future traffic conditions can be especially useful for the proactive adaptation of signal plans (Chapter 6) and for vehicular route guidance (Chapter 7).

Forecasts are just one way to make a technical system more resilient. The use of machine learning can lower the reaction and adaptation times, while leading to better strategies and consequently, a higher system performance. Furthermore, machine learning enables the traffic control system to react autonomously without human interaction. We will learn more about this approach in chapters 4 and 8.



(a) Elman recurrent network with five inputs, ten hidden neurons, and three outputs.

(b) Forecasts (blue dotted line) and actual traffic flow curve (grey solid line).

Figure 2.7: Single-step forecasting with an artificial neural network for an exemplary Monday traffic flow. The forecast represents the estimated traffic flow five minutes into the future.

The following paragraphs discuss new concepts to make the self-optimising traffic control system OTC more resilient. Therefore, these are the grounding features of this newly developed resilient traffic management system.

1) Improving signalisation behaviour with forecasts: Due to safety reasons and to allow the current signalisation plan to show its performance, a review of the current control strategy takes place at the begin of each second or third cycle. Taking a standard duration of such a phase plan of about 90 seconds into account, OTC decides about an adaptation in intervals of four to five minutes. Hence, an appropriate forecast would alter the adaptation strategy of OTC: Instead of selecting the most promising signal plan for the current traffic conditions, selecting the one for the estimated traffic demands in the near future can lead to a better anticipatory adaptation strategy. Historical and current traffic flow of turning movements are used to estimate the most probable traffic state at a certain future point in time. This knowledge can be taken into account when proactively adapting the controller's strategy. Here, the forecasted values are integrated into the signalisation and coordination strategies [SH16]. The focus lies on an isolated intersection controller incorporating forecasts based on its local sensory data. Chapter 6 presents this approach in more detail.

2) Proactive rule generation: Additionally, the generated forecasts can be used to proactively extend the rule-base of the XCS at layer 1. Therefore, layer 2 could be triggered if the rule base of the learning component situated at layer 1 does not contain an appropriate classifier for the forecasted situation. In this context, "appropriate" means either no matching rule or only matching rules with insufficient performances. However, this approach is not considered within this thesis. The interested reader is referred to *Stein et al.* [SER+16] in terms of niche exploration, knowledge discovery, and knowledge interpolation techniques in learning classifier systems.

3) Anticipatory route guidance: As described later in chapter 7, OTC's dynamic route guidance mechanism works on the principle of proposing route recommendations to drivers by variable message signs before intersections. This can result in frequently changing routes which might affect the acceptance of guidance by drivers. In addition, route guidance only takes into account the current traffic state. Hence, neither the impact of providing route recommendations to drivers, nor the traffic conditions being present at the next intersection (when the driver finally arrives there) are considered. Therefore, short-term forecasts of traffic flow is used to generate time-dependent route recommendations that define a more sophisticated time-varying representation of the traffic conditions while traversing the network. By using the information, the upcoming changes serve as a basis to investigate a resilient route guidance system. The resilience of the route guidance system is characterised by avoiding oscillating behaviour (in this context, oscillating behaviour means alternating route recommendations) and identifying shortages in capacities before they occur.

4) Automatic congestion detection: Existing congestion detection algorithms analyse data gathered by distributed road sensors and try to classify the traffic patterns into congested or uncongested. These detection methods often rely on static decision trees or previously calibrated models. In contrast, self-adaptive machine learning techniques do not rely on complicated, time-consuming calibration at design time. They improve their performance and knowledge base at runtime by sophisticated reinforcement of their internal model. This helps to reduce the negative impacts of incidents and to shorten the duration or to even prevent the occurrence of congestion. Consequently, the road network becomes more robust against disturbances. For more details, see chapter 8 and chapter 9.

2.4 Resilience through forecasts: Module for time series forecasting

Forecasting of time series is one precondition for introducing resilience into technical systems. In the remainder of this chapter, the architecture of the newly developed module for forecasting of univariate time series is presented. As depicted in figure 2.4, this module extends the standard observer/controller architecture. In certain time intervals, sensor values from real detectors or from a simulation environment are passed to the OTC system. Figure 2.8 presents the process. The OTC Manager acts as adapter between external inputs and the internal modules. Afterwards, the received values are given to the observer at layer 1 and finally to the forecast module where they are stored as time series and used for forecasting.

However, due to its general architecture, the module can also be used in a stand-alone mode. The difference between these two modes is how the data is received. As part of the observer, input values are received one by one in certain time intervals, whereas in stand-alone mode, a whole time series can be consumed at once. For more information on this mode, see section 2.4.3.

The module is based on the ideas of multi-model ensemble learning [Arm01]. It refers to the combination of a set of forecast methods. Several forecast methods independently create forecasts based on their individual model which are then combined by a combination strategy. The applicable strategies range from the simple average of the forecasts to sophisticated machine

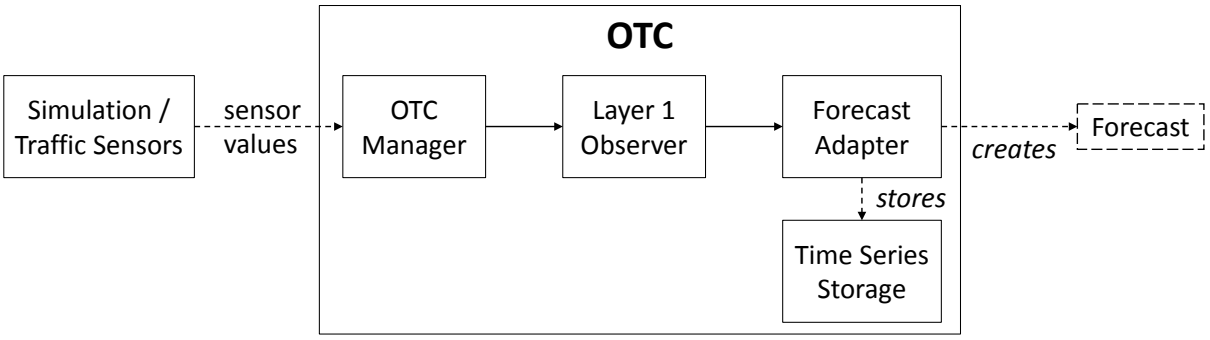


Figure 2.8: Standard work flow of the forecasting framework in OTC.

learning algorithms which learn the best combination at runtime (i.e. learning classifier systems [STHA15]). Finally, the combined forecast is returned.

2.4.1 Architecture

Figure 2.9 depicts the architecture of the forecast module in more detail. The *ForecastModule* serves as an adapter to the interior modules, hiding the complexity of the underlying framework from the end user. Most significantly, other software modules can be easily extended by this module, thereby acquiring the capability of time series forecasting. The *ForecastModule* knows the available *ForecastMethods* and *CombinationStrategies*, and is responsible for the initialisation of those selected by the user. The user only has to specify which of the provided forecast methods he wants to be activated and which combination strategy should be used to combine the forecasts (Configuration in external property files). In case a forecast is demanded, the *ForecastModule* triggers the activated *ForecastMethods*. Each *ForecastMethod* creates a forecast based on its individual model and configuration, and returns the result. This set of forecasts is then passed on to the *CombinationModule*, where they are combined based on the selected *CombinationStrategy*, and their individual weights. Finally, the combined forecast is returned by the *ForecastModule*. Additionally, the *ForecastMethodEvaluator* stores the forecasts and the according actual values. These forecast-actual value pairs can later be used for plotting and provide input to the forecast error measures.

2.4.2 The time series forecasting process

In the following, each step of the time series forecasting process as introduced in this work is explained. The sequence diagram in figure 2.10 illustrates the interaction process between the individual modules of the forecast component when a new combined forecasts is demanded. The diagram shows the lifeline of each entity as a vertical line and the arrows display the time-ordered exchange of messages between different entities. The process works as follows. First, an external entity triggers the creation of a forecast. This initial message is retrieved by the *ForecastAdapter*. This module triggers the *ForecastModule*. For each method *fm* from the set of chosen forecast methods, the method *getForecast()* is called. Each generated forecast is returned

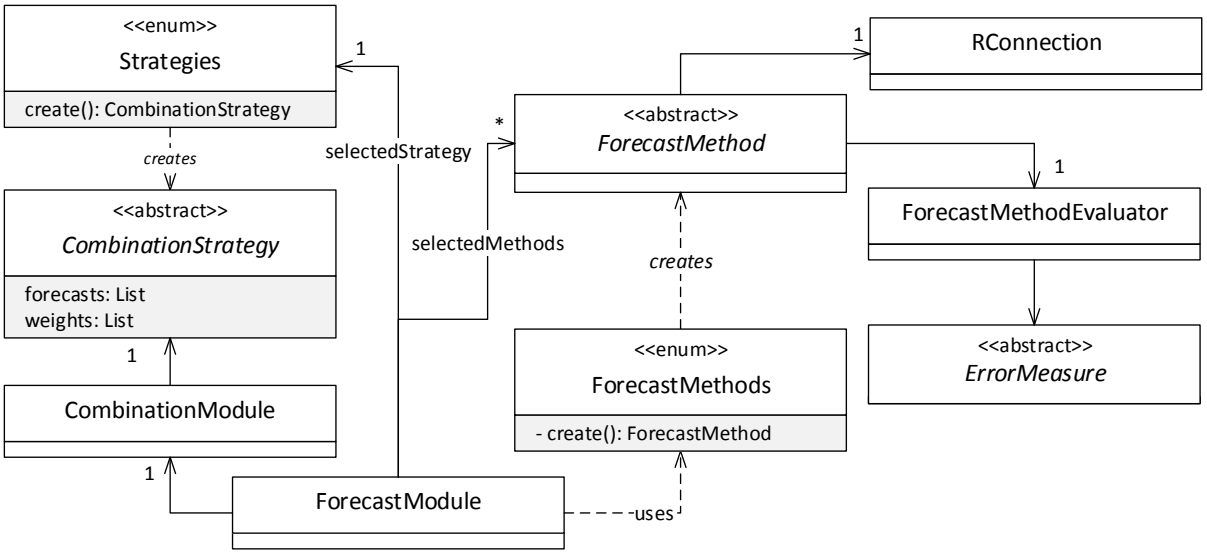


Figure 2.9: Schematic view showing the main components of the forecast component.

and added to the vector of all forecasts. Afterwards, depending on the selected combination strategy, the weights for the combination process are computed. The weights and the forecasts are forwarded to the *CombinationModule* which computes the linear combination of the forecasts. Finally, the combined forecast is returned.

2.4.3 Stand-alone mode

In stand-alone mode, the module receives a complete time series as input and creates forecasts for each data point. The typical workflow of the stand-alone mode is shown in figure 2.11. In the following, a short description of each step of the process is given.

Time series analysis and pre-processing: Initially, a time series is put into the module. The framework offers tests for certain time series characteristics (i.e. trend, seasonality, skewness, and stationarity) which can be executed automatically. Their respective results are stored for later evaluation. Furthermore, the time series can be pre-processed. Several techniques are available to normalise the time series to a certain value range. Most real-world time series, such as in economics, are not stationary and exhibit trends or seasonal characteristics. In case the time series is non-stationary, it can be converted into a stationary representation. Otherwise, the resulting forecasts can have a bias due to the non-stationarity. Alternatively, an ARIMA(p,1,q) or ARIMA(p,2,q) forecast model can be used to differentiate the time series and therefore remove the non-stationarity.

Automatic selection of forecast methods: Based on the previous time series analysis and pre-processing, certain specific forecast methods can be activated. In case the time series exhibits seasonal patterns, a seasonal forecast method, such as SARIMA could be beneficial to choose. Based on heuristics, the module could make this selection on its own. To keep it simple, this

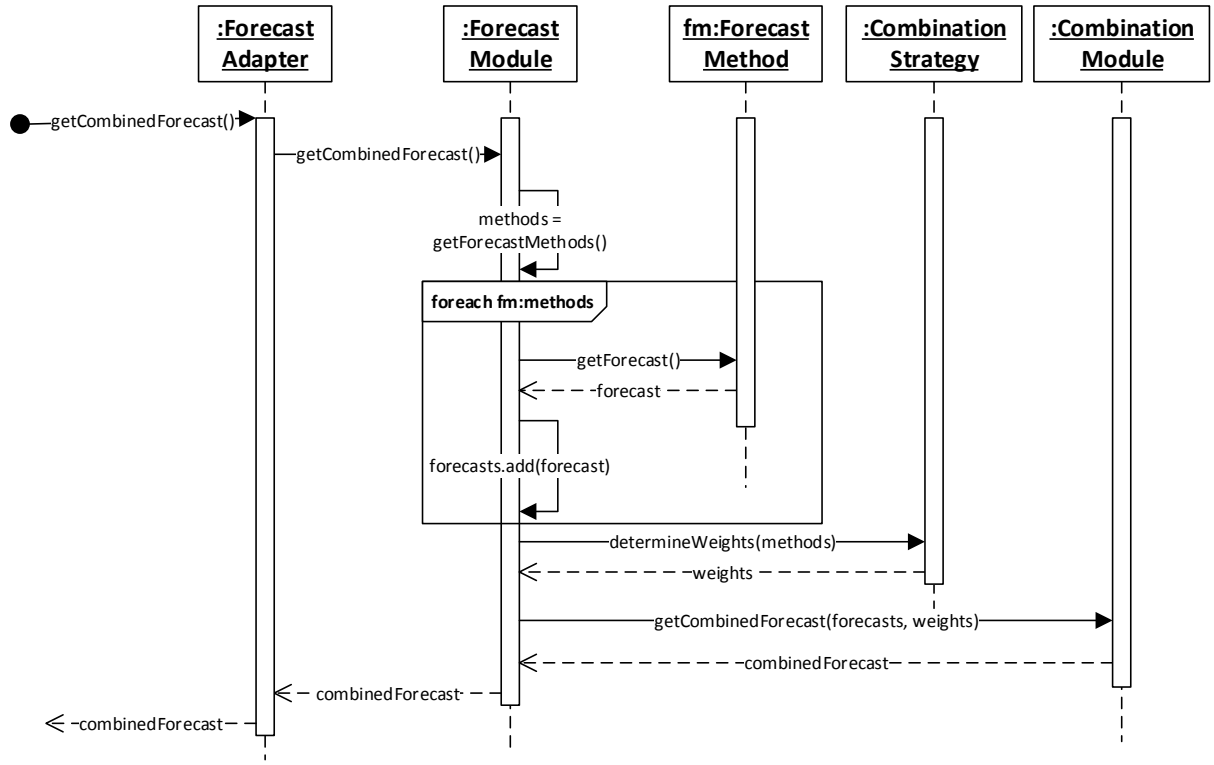


Figure 2.10: The sequence diagram depicts the process for retrieving a combined forecast.

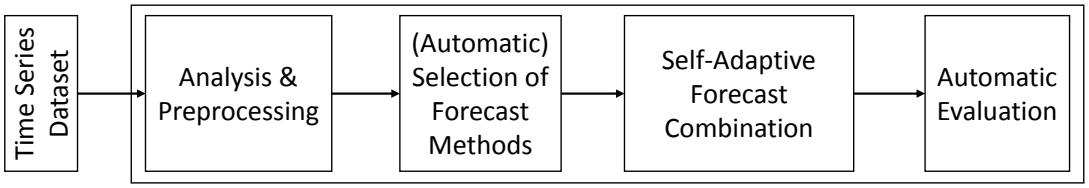


Figure 2.11: Standard work flow of the forecasting framework in stand-alone mode.

functionality was not included into the forecast module. The forecast methods are selected by the user by modifying an external configuration file.

Self-adaptive forecast combination: The forecast module relies on R [R C15], a sophisticated framework for statistical computing. R offers several packages for time series analysis, forecast performance measures, and forecasting methods. The forecast methods are used as provided by the *forecast* package for R [HK08] to generate forecasts for univariate time series. Rserve [Urb03] couples the Java implementation with the R backend by establishing a local TCP/IP server. The framework follows a modular design to allow easy integration of additional forecast methods, forecast combination strategies, and forecast error measures besides R. The term self-adaptive refers to the possibility to use dynamic forecast combination strategies that calculate time-varying weights according to the individual performances of the selected forecast models.

Automatic evaluation: Finally, several evaluation measures can be calculated automatically whereas their results are exported to log files. Beyond others, the measures include RMSE, Theil's U-statistic, MASE, and SMAPE. Additional measures can be added easily. Furthermore, plots can be automatically created for further analysis. Again, R is used for the generation of plots, for example histograms of the forecast error distribution and box plots of the error measures.

2.5 Summary

In this chapter, the fundamentals of traffic control were depicted and the self-organising Organic Traffic Control system was introduced. It is based on a multi-layered architecture following concepts from the Organic Computing domain. Based on several definitions for resilience, the concluding definition "proactive robustness" will be used throughout the remainder of the thesis. Thus, proactiveness is key to a more resilient system [LH05]. In terms of organic systems, this will be achieved by including a forecast module for univariate time series into the observer component.

Steps to extend the reactive Organic Traffic Control system with anticipatory control strategies were outlined and new features to improve the robustness of the road network were discussed. These features outline the remainder of this thesis. Finally, the forecast module extending the existing observer architecture was presented. This module will play an important role for two contributions of this work: anticipatory signalisation and proactive route guidance.

Chapter 3

Forecasting of univariate time series

There is a large body of literature on forecasting of time series, a field of research that has been investigated for decades (see [AA13]). The aim of time series analysis is to understand the dependencies between the observed values of a series (e.g. time-ordered values from a sensor monitored in certain intervals) and to fit an adequate model representing its inherent structure. With the help of this model, predictions about future developments can be derived, i.e. to make forecasts. Therefore, it is an essential part of the theoretical framework of this thesis.

A lot of researchers have proposed various time series forecasting models and methods to improve their forecasting accuracy. There is a vast amount of literature on heuristics, statistical models, parametric and non-parametric methods, and machine learning techniques to forecast time series. Still, there is no method that outperforms all others in every scenario or domain [Cle89]. The most popular class of time series models was presented by *Box* and *Jenkins* in 1976: the autoregressive integrated moving average (ARIMA) [BJ76]. Basically, ARIMA depicts a stochastic process modelling framework of parametric regression models building on the assumption that a time series is linear and can be described through a statistical distribution.

This chapter gives an overview of approaches for the time series forecasting problem and discusses the benefit of novel meta-learning approaches based on machine learning techniques. Many of the recently published papers and comparative studies [AAGES10] deal with the application of machine learning techniques. Artificial neural networks [HOR96, ZEPYH98] and support vector machines [Cao03] have shown to be powerful models for time series modelling and forecasting, often outperforming classic statistical models. Their drawbacks are their need for more computational power and their internal complexity since the representation of the evolved models is not easily interpretable by humans.

The remainder of this chapter is organised as follows. First, section 3.1 introduces the term time series and further important terms and definitions in section 3.2. Afterwards, an overview of the related work in the field of time series forecasting is given in section 3.3. A new forecast module for time series forecasting is introduced. Finally, a detailed look is spent on strategies for the combination of several forecast methods in section 3.4 and on some forecast performance measures in section 3.5.

3.1 Terms and definitions

In general, a time series describes a time-ordered sequence of data points $X_1, ..., X_t$, derived from a system in discrete time intervals of successive measurements [AA13]. These data points are random variables where X_1 denotes the value at the first point in time, X_2 the second, and so on. If the time series only contains records of a single variable, it is called **univariate**. It is **multivariate** if more than one variable is recorded. A time series can be **discrete** or **continuous**. If an observation is raised at every instance of time, it is continuous, whereas a discrete time series contains only values measured at discrete points in time. Stock prices, monthly sales, and annually population are just a few examples of time series from real-world applications. Usually, these time series are recorded in fixed time intervals, such as daily, monthly or yearly. In the following, the focus lies on univariate, discrete time series.

Time series can also be defined by a *function* $f(x)$ with an independent variable x derived from a complex system. However, usually this function can not be described exactly. Therefore, future trends are estimated by observing previous behaviour. This is accompanied by the establishment of a *mathematical model* based on past sample data describing the time series. Forecasts derived from this model are often used for decision making in real-world applications, such as the stock market or traffic management [STH14, STH15b]. Thus, making accurate and reliable forecasts based on a suitable model is of great importance for the performance of these systems.

3.2 Time series decomposition

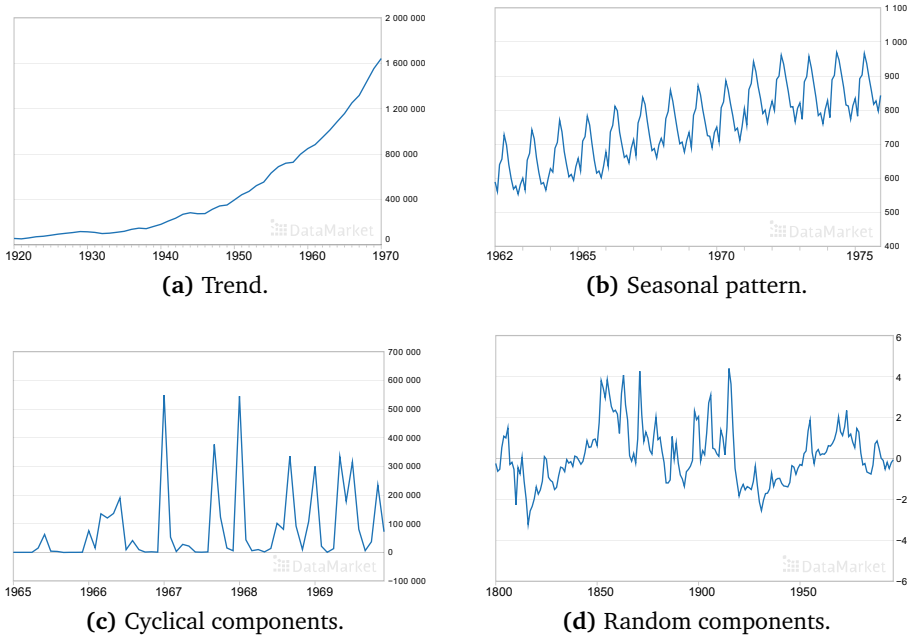
Typically, time series can be divided into four components. A time series can have a **trend**, be **cyclical**, can exhibit **seasonal** behaviour, or can have **irregular** components (see figure 3.1 for examples) [KW08]. The presence of any of these components can usually be detected by simply looking at time plots of the time series or by statistical evaluation. In the following, these four components are briefly described.

Trend: A trend is a long-term increase or decrease of the level of a time series. Figure 3.1a shows the increasing total electricity consumption (a positive trend) in the U.S. from 1920 to 1970 in kilowatt-hours (millions).

Seasonality: Seasonality is defined by seasonal factors with a fixed period of time, e.g. the yearly recorded temperature. This is usually determined by repetitive ups and downs in the time series graph. Figure 3.1b presents a time series of the monthly milk production in pounds per cow from January 1962 to December 1975. Besides its seasonal behaviour in yearly intervals, it is also exhibiting an increasing trend.

Cyclical component: A time series has a cyclic component if it exhibits fluctuations with no fixed period. The time series in figure 3.1c illustrates a cyclic pattern for the number of Harmon foods consumer packs from January 1965 to December 1969.

Random components: At last, irregular or random components are difficult to model since they are caused by unpredictable influences. It can be seen from Figure 3.1d that the annual copper prices (1800-1997) exhibit such irregular behaviour.



(a) Trend. **(b)** Seasonal pattern.

(c) Cyclical components. **(d)** Random components.

Figure 3.1: The time plots show exemplary time series exhibiting different patterns and characteristics.[1]

Stationarity: Besides the previously mentioned components, a time series can be *stationary* or *non-stationary*. A process or a time series is *stationary* when the random mechanism producing this process does not change over time [GR53]. Formally speaking, a process x_t is *strictly stationary* if the stochastic variables $x_{t_1}, x_{t_2}, \ldots, x_{t_n}$ have the same joint probability distribution as $x_{t_{1+k}}, x_{t_{2+k}}, \ldots, x_{t_{n+k}}$ for all n, $t_1, t_2, \ldots, t_n$ and every k. The strict definition of stationarity states that the marginal distribution of a time series process does not change over time. As this definition is generally to strict for everyday processes, a weaker assertion only demands that the mean and the covariance stay stationary over time (*weak stationarity*) [KW08]. To mathematically test a time series for stationarity, so-called unit root tests are used. The (Augmented) Dickey-Fuller [DF79] and the Phillips-Perron [Per88] tests check a time series for non-stationarity. They distinguish between trend stationarity and difference stationarity using the existence of a unit root as the null hypothesis.

Mostly, real-world time series, such as in economics or transportation, are not stationary and exhibit trends or seasonal characteristics. Non-stationary data can not be modelled accurately and is therefore difficult to forecast as the mean, variance, and covariances may vary over time. At least, the resulting forecasts may have a bias deriving from the non-stationarity. However, non-stationary data can be converted into a stationary representation by trend elimination, by removal of the seasonal factors, or by a k-th order differentiation with a lag-operator [KW08].

[1] Time plots taken from datamarket.com

As we have seen, time series can exhibit several different characteristics. For decades, researchers try to find improved techniques to model and to forecast time series more accurately. In the following, a brief overview over the field of time series forecasting techniques is given.

3.3 Methods for time series forecasting

As stated previously, time series forecasting will play an important role in this work for the later contributions. Since the development of new forecast methods for time series is out of the scope of this thesis, we will rely on established forecast techniques. The books by *Kirchgässner* [KW08] and by *Adhikari* [AA13] provide a good introduction to time series forecasting in general. An extensive overview over the state of the art of forecast methods for traffic flow is given by *Bolshinsky and Freidman* [BF12]. In the following, time series models are categorised as qualitative and quantitative approaches, parametric and non-parametric regression, machine learning algorithms, and ensemble forecasting techniques.

3.3.1 Qualitative vs. quantitative approaches

Qualitative forecast methods are typically used when historical data is sparse or not available. In this case, opinions and judgements from experts are considered. Consequently, these forecasts are subjective. One well known representative is the Delphi method where experts give their judgements in an interactive, round-based fashion until a consensus is reached [Sek15, RW99]. In contrast, quantitative methods rely on historical data from which they construct a forecasting model to do forecasts. Their advantage over qualitative methods is that they are objective. Their disadvantage is that they need training data and a suitable model describing the underlying process. In the following, some quantitative approaches are explained in more detail.

3.3.2 Parametric regression

Several approaches try to explain the correlation between adjacent points in time. The time domain approach tries to model future points of a time series as a parametric function of the current and previous values. Parametric regression is the search for the coefficients of a polynomial function which minimises the sum of the quadratic errors for a given set of training data [Alp08].

ARIMA models

Box and *Jenkins* [BJ76] developed a stochastic process modelling framework of parametric regression models called ARIMA (**A**uto **R**egressive **I**ntegrated **M**oving **A**verage). *Shumway* and *Stoffer* [SS06] give an introduction to time series analysis, and how ARIMA processes can be used for time series forecasting. An ARIMA model can be divided into three main components: the *autoregressive* (AR), the *integrative* (I), and the *moving average* (MA) part. The model is defined by the parameters p, d, and q, where

- **p** is the number of autoregressive terms,
- **d** is the number of non-seasonal differences needed for the transformation into a stationary stochastic process, and
- **q** is the number of lagged forecast errors.

The ARIMA model is then denoted as $ARIMA(p, d, q)$, with p, d, and q being non-negative integers. The autoregressive (AR) part is based on the assumption that the next value x_{t+1} of a time series can be expressed through its previous values $x_t, x_{t-1}, \ldots, x_{t-k}$ and a Gaussian distributed white noise e_t with mean zero and variance σ^2 by the formula

$$x_t = \theta_1 x_{t-1} + \theta_2 x_{t-2} + \cdots + \theta_k x_{t-k} + e_t. \tag{3.1}$$

ARIMA models can also be used for non-stationary time series, where an initial differentiation step is applied (the integrative part I) [Chr05]. The integrative part tries to transform a non-stationary time series into a stationary representation by differentiation with a backward shift operator B where

$$BX_t = X_{t-1}, \forall t > 1. \tag{3.2}$$

The first difference operator Δ is then expressed as

$$\Delta X_t = (1 - B)X_t. \tag{3.3}$$

This generalises to the i-th difference as

$$\Delta^i X_t = (1 - B)^i X_t. \tag{3.4}$$

In case the time series is stationary, no differencing is needed. One order differencing assumes that the original series has a constant average trend. Two orders of differencing assume that the original series has a time-varying trend. At last, the MA term models a moving average process, considering the average of the last few observations to filter out noise.

Special cases of ARIMA models

Many other algorithms can be expressed by an ARIMA(p,d,q) model. ARIMA(0,1,0) with $y_t = y_{t-1} + \epsilon_t$ is known as *random walk*. It is often used for non-stationary time series, such as economic and stock prices series [AA13]. The moving average is equal to an ARIMA(0,0,1) model. Another common forecasting model is the simple exponential smoothing model (ARIMA(0,1,1)) [Bro56]. Based on a smoothing parameter α with $0 \leq \alpha \leq 1$, the smoothed value s_t is calculated recursively as

$$X_{t+1} = \begin{cases} s_1 = X_1, & \text{for the first value} \\ s_t = \alpha X_t + (1 - \alpha)s_{t-1}, & \text{otherwise.} \end{cases} \tag{3.5}$$

The double exponential smoothing model [Hol04] extends the previous model, also considering trend patterns. It equals to ARIMA(0,2,2). The variable b is introduced as an estimate for the trend at time t. The smoothed value s and the trend factor b are initialised by

$$\begin{aligned} s_1 &= X_1. \\ b_1 &= X_1 - X_0. \end{aligned} \tag{3.6}$$

Then, a single-step forecast is calculated as

$$\begin{aligned} s_t &= \alpha X_t + (1-\alpha)(s_{t-1} + b_{t-1}). \\ b_t &= \beta(s_t - s_{t-1}) + (1-\beta)B_{t-1}. \end{aligned} \tag{3.7}$$

whereas a multi-step forecast of m steps into the future is given by the formula

$$X_{t+m} = s_t + mb_t. \tag{3.8}$$

The ARIMA model can be extended by a seasonal component, whereas it is then called seasonal ARIMA (SARIMA). SARIMA is able to deal with seasonal non-stationary and seasonal data as well. An additional seasonal differentiation removes the seasonal non-stationarity from the time series. The model is defined by $ARIMA(p,d,q,) \times (P,D,Q)$, where P is the number of seasonal autoregressive terms, D is the number of seasonal differences, and Q is the number of seasonal moving average terms.

For more examples, the interested reader is referred to [Tho80].

3.3.3 Non-parametric regression

In the context of forecasting, non-parametric regression estimation has several benefits over parametric methods [VFC07]. The approach is more flexible and well adaptable to local features, and multi-step forecasts are easily raised, but with more computational effort. For a successful application, more data compared to parametric regression is needed. In contrast to parametric approaches, where we search for the coefficients of a polynomial, the non-parametric methods do not rely on that assumption, but have to compute a function $r^t = g(x^t) + \epsilon$ for a given set of data $X = \{x^t, r^t\}$ with $r^t \in \mathbb{R}$ [Alp08]. Under the assumption that neighbouring points of a time series exhibit equal behaviour for $g(x)$, this calculation can be done by local linear regression models [SLX+03]. Compared to methods of this class, ARIMA models showed to be statistically superior [ZL03].

Typical representatives of non-parametric regression methods are k-nearest neighbour methods [AQC13], kernel methods (e.g. support vector machines and Gaussian process regression [RPHR07]), and spline smoothing [HS04]. *Härdle* [Här90] gives an extensive introduction into the fundamentals of non-parametric regression. He states that non-parametric approaches provide a versatile method to explore the relationship between two variables. They need no fixed parametric model and deal well with missing values by interpolating between adjacent points.

3.3.4 Machine learning algorithms

Another class of forecast techniques follows the concept of machine learning. *Alpaydin* [Alp08] defines machine learning as the optimisation of the parameters defining a certain model. This model is optimised according to sample data or past experience, measured after defined optimisation criteria. A model can be *predictive* to forecast certain behaviour, or *descriptive* to gain knowledge about the data, or both. Typical use cases are classification tasks and regression problems. The learning process can be *supervised*, which means that the model learns the relationship between input and output, whereas the output values are known. In contrast, with *unsupervised* learning, the output values are unknown and the goal is to find patterns in the input data through density estimation or clustering techniques. Time series forecasting is typically supervised as the model has to be trained beforehand.

Comprehensive comparison studies over machine learning techniques for time series forecasting in general are presented by [AAGES10, BBTLB13]. In terms of traffic flow forecasting, many different methods are applied. Some researches make use of Bayesian networks ([SZZ05, CMS08]), kalman filters [OS84], or support vector machines [Mar02, Cao03]. Artificial neural networks (ANN) [Gur97] are widely used algorithms, also in the field of time series forecasting. ANNs are parallel, distributed computational models that have the ability to learn, to generalise, and to cluster data. They are also applied successfully to the forecasting of traffic flow [STH13b, CDSC12, STH15a].

In conclusion, a large amount of research deals with the application of single methods. As we have seen, there exists a large number of different approaches for forecasting of time series. Since all of them have their advantages and disadvantages, its difficult to make a selection. Usually, expert knowledge is needed to make an informed decision. Therefore, we have to evaluate other solutions, such as ensemble forecasting.

3.4 Ensemble forecasting: Combining forecasts

Ensemble forecasting attempts to enhance the forecast accuracy by using multiple models instead of only relying on a single model. In the following, an introduction to this field of research is given.

In 1969, *Bates* and *Granger* [BG69] are the first to suggest a combination of independent forecast methods. They use the past forecast errors and the error variances to determine a weighted combination of two forecast methods. Their combined approach is the starting point for many other researchers in this field. The combination of several individual forecast models is also referred to as *ensemble forecasting* [Arm01]. A number of reviews focus on the combination of, and selection among several forecast methods [TWX+09, Arm01, Cle89].

In a recent study, *Adhikari* and *Agrawal* [AA14] investigate the strengths and weaknesses of several linear combination strategies in comparison to individual forecast techniques. Their results indicate that the forecast accuracies of the single forecast methods vary notably, and that all combination methods significantly reduce the forecast error. The findings of *Hibon* and *Evgeniou* [HE05] state that the worst performance among individual forecast methods is

significantly worse than the worst combination. Still, the best individual method and the best combination perform on average on a similar performance level. In fact, there exists a theoretical proof that a well selected and parametrised ensemble predictor has smaller squared forecast errors compared to its base predictors. *Larrick* and *Soll* [LS03] demonstrate that combinations can lead to worse performance in special cases. However, this is not statistical relevant. In conclusion, we deem ensemble forecasting as the most promising approach.

While the combination of forecasts derived from an ensemble of several forecast methods has been studied since 1969, the use of machine learning approaches to combine several forecasts has first gained interest since the early 1990s. A recent survey summarises state-of-the-art ensemble methods, such as bagging, boosting, random forests, decomposition methods, and many more [RZS16]. In order to improve the reliability of forecasts, and to overcome the limitations and drawbacks of individual techniques, different approaches have been discussed in literature. These include among others,

- finding the best individual model from a set of forecast methods [HHKA13, PL04],
- combining the forecasts from a given set of methods [Cle89, WM83],
- and finding the optimal set of candidate methods that most improves the forecast accuracy [Alp08].

Kucheva [Kun04] introduces several combination schemes and discusses how to chose the best subset from a given set of forecast methods. Still, the performance of the combination is likely to not improve further, when more and more methods are combined. In fact, the degree of saturation in the accuracy is assumed at four to six methods [Arm01]. The combination of several forecast methods is successfully used to improve the forecast accuracy in several real-world domains [RZS16, STH14]. In the following, the second approach is pursued, trying to find the optimal combination of several independent forecasts from a fixed set of forecast methods.

The remainder of this chapter is as follows: First, a formal representation of the forecast combination problem is given. Based on these theoretical ideas, the standard combination process is presented. We move on, summarising guidelines for the combination of forecasts. Then, several state-of-the-art combination strategies are presented. Afterwards, the framework for time series forecasting based on forecast ensembles is introduced which will be used later on to make short-term forecasts of traffic flow. At last, some important forecast performance measures are summarised.

3.4.1 Problem formulation

Figure 3.2 illustrates the standard process for the combination of several forecasts. First, several forecasting methods P_1 to P_n independently generate forecasts based on their individual model, and historic and recently monitored data. Their forecasts F_1 to F_n are then combined by a combination strategy executed by the *Forecast Combiner*. The according weights can be calculated by a simple approach, such as assigning equal weights to all models, or by a machine learning technique that learns the optimal weights over time. Based on historic time series data $X^{t-1} \dots X^{t-m}$, and the history of previous forecasts of each forecast technique $F^{t-1} \dots F^{t-k}$,

the *Weight Learner* derives the weights according to the chosen strategy. Finally, the combined forecast $\hat{F}$ is calculated based on these weights and the respective forecasts.

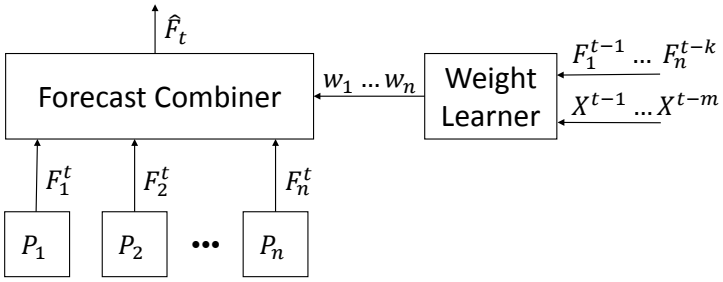


Figure 3.2: Weighted combination of forecasts F_1 to F_n created by multiple forecasting techniques P_1 to P_n.

The combination of several forecast models with a machine learning technique is known as *stacked generalisation* [Wol92]. Each individual forecast method is seen as a *base learner* which tries to learn the underlying model representing a certain time series. Based on its internal model, each learner gives its independent *vote* F_i (or forecast in terms of time series forecasting). For a combined vote, the independent votes are then combined into one comprehensive result. The concept of stacking is illustrated in figure 3.3. The output $F_1, F_2, \dots, F_n$ of the base-learners $P_1, P_2, \dots, P_n$ is combined with a learning system $f()$. This combination is not restricted to be linear. The meta-learner $f()$ learns the correct output for a certain input combination at runtime. This component estimates and corrects errors made by the base learners in certain situations. A concrete example is presented in section 5.2, where a learning classifier system is applied as meta-learner.

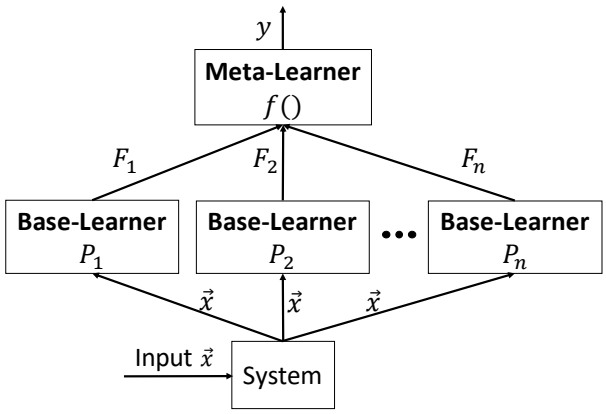


Figure 3.3: Combining several forecasts with stacked generalisation (based on [Alp08]).

In general, the forecast combination problem can be expressed as finding a vector of the optimal weights for an input of forecasts from different forecasting models to obtain the optimal combination of these forecasts. Concluding, the forecast combination problem can be formulated as follows: Given a time series, what are the optimal weights in a certain time step, to obtain the optimal combination of forecasts from several independent forecast methods?

The easiest way to combine multiple base learners is by a linear combination, also known as *voting*. This concept was first used for classification tasks where the winning class is the one with the most votes. It can be easily transferred to regression problems and to the combination of forecasts. The simple average calculates the average of the forecasts F_i for time step $t+k$ delivered by a learner i for $i \in \{1 \dots n\}$ by

$$y(t+k) = \frac{F_1 + F_2 + \ldots + F_n}{n}. \tag{3.9}$$

Thereby, y is the combined forecast value, and F_1 to F_n are the forecasts provided by n different forecast methods. Another simple and fast approach is to take the median of all forecasts. Its disadvantage is that if one or more forecasts are biased, the median will equally tend to be too high or too low. Alternatively, the output of each individual forecast technique $F_1, \ldots, F_n$ is considered with varying impact and weighted according to the expected accuracy. Let $W_i (i = 1 \dots n)$ be the weight of forecaster i, and F_i its respective forecast. Then, the combined forecast $y(t+k)$ for the time step $t+k$ can be calculated as a linear combination by

$$y(t+k) = f(F_1, \ldots, F_n, W_1, \ldots, W_n) = \frac{\sum_{i=1}^{n} F_i * W_i}{\sum_{i=1}^{n} W_i} \tag{3.10}$$

where F_i is the forecast value and W_i its assigned weight with $W_i \geq 0, \forall i$. Usually, the value range of each weight is restricted to $[0, 1]$, and the sum of all weights equals 1. Equal weights for all learners consider all forecasts as equally important.

3.4.2 Guidelines for the ensemble forecasting process

Before an ensemble-based forecast can be created, several decisions have to be made. First, a optimal set of forecast methods has to be chosen. This is especially difficult for users which are new to the field of forecasting. *Wang* [WSMH09] recommends rules for the forecasting method selection problem. Second, the optimal set of weights has to be found so that the combination of the individual forecasts leads to better results then taking a single forecast method or any of its subsets. In other words, the forecast error and its variance are to be minimised for a given time series. The disadvantage of a combined learner is its lack of interpretability due to its complexity.

The following simple rules by *Alpaydin* [Alp08] represent four essential rules on how to select and train base learners to improve the forecasting performance:

- **Select different algorithms**, such as parametrisable and non-parametrisable methods. Even if these base learners are trained on the same data sets, each model will have another representation of the data which therefore leads to different results.
- **Choose different hyper-parameters** when using the same method more than once. A hyper-parameter is a supplemental parameter which is, for instance, a parameter defining the configuration of the underlying parametric model. Examples are the number of hidden neurons of an ANN or the number of neighbours in nearest neighbour algorithms.
- **Use sensor data from different sources** as they represent the problem more precisely. For instance, the use of a frontal and a side photo of a face can reduce the detection error

in face recognition software [KHDM98]. Each base learner utilises another data source, whereas the combination of the individual decisions of these learners leads to a multi-dimensional view.

- **Train each base learner on different training sets** (all representing the same problem domain). Therefore, the problem of overfitting the model of an individual method is mitigated. These training sets can either be drawn randomly which is known as *bagging*. Or the base learners can be trained sequentially which is called *boosting* [AA13].

In accordance to the third point, the combination of several learners in a large learning model trained on the whole training set can lead to overfitting. The combination of several learners which are trained on different parts of the training set or on different training sets avoids this problem. Accordingly, *Ren et al.* [RZS16] state that a good ensemble is designed based on the principles of data diversity, parameter diversity, and structural diversity. In consequence, the focus lies on the diversity of the generated models. However, these rules do not help to achieve a valuable combination of several forecasts. *Armstrong* [Arm01] proposes seven practical guidelines when combining forecasts:

- **Use data from different sources** or different methods to adjust for bias.
- **Use at least five different forecast methods** to get the optimal improvement in terms of forecast accuracy.
- **Use formal procedures** which are well documented and executed automatically instead of judgemental weighting.
- **Use equal weights** if there is no strong evidence for unequal weighting.
- **Use trimmed means** to get rid of extreme forecasts when there are at least five forecasts.
- **Assert higher weights** to methods that have been proven to perform better.
- **Use domain knowledge** to adapt the weights.

These guidelines give a good intuition how to select a high quality subset of base learners from a set of available learners. Furthermore, they indicate how to chose the weights for the linear combination of forecasts. Still, they do not answer the question how the optimal weights can be derived. The following section deals with strategies for the linear combination of a set of forecasts, describing how these strategies find the according optimal set of weights.

3.4.3 Combination strategies

Over the past few decades, a number of authors have suggested a notable amount of approaches to solve the forecast combination problem [Cle89, Arm01]. These concepts range from simple statistical ensembles, such as the näive simple average or the trimmed mean, to more evolved methods which rely on statistical observations or machine learning models. These strategies are use a weighted linear or non-linear combination of several individual forecasts based on constant weights. The problem with constant weights is that the optimal weights are not fixed for a certain time series and a given set of learners, but change over time depending on the most recent data points and characteristics of the time series. So, it can be advantageous to

use time-varying weights, and to update the weights repeatedly after certain intervals of time. So far, only few researchers have proposed time-varying methods for the forecast combination problem [DGT94, ZY04, STHA15, AV16]. Other methods switch dynamically between several combination strategies, based on the number of methods generating the forecasts [dMBT00]. In accordance with *Adhikari* [AA14], linear forecast combination strategies are classified as follows:

Statistical methods

The naive simple average, the median, and the trimmed mean [JW08] (the highest and lowest forecasts are removed) represent statistical methods which are easy to understand and fast to compute. Although it offers suboptimal weights, the simple average (the average of the forecasts) is often superior to more sophisticated methods. Still, their major drawback is their sensitivity to extreme values. Therefore, if the forecasts are biased, the combined forecast is likely to be biased as well.

Error-based methods

Error-based methods weight the forecast of each forecasting method based on its accuracy derived from a training set. Therefore, a given time series is divided into a training and a validation set. The performance is then evaluated on the training set with an error measure, such as the mean absolute error or root mean square error. The respective weight is taken as inversely proportional to this forecast error [Arm01]. These strategies and their respective benefits and drawbacks are briefly summarised in [AA13].

One representative called *optimal weights* [Dic73] calculates the vector w of weights to minimise the error variance of the combination of n forecasts as follows:

$$w = M_V^{-1} I_n (I_n^{'} M_V^{-1} I_n)^{-1} \tag{3.11}$$

where I_n is the $n \times 1$ matrix with all elements set to 1 and M_V resembles the covariance matrix of the forecast errors. This $n \times n$ matrix (with n being the number of forecast methods) contains the covariances between the forecast errors of all forecast methods at a particular point in time (e.g. $\sum_{i,j}$ is the covariance between the absolute error of forecast i and forecast j). It is assumed that there is no correlation between two different forecast errors. Other restrictions which can be included are that just the individual forecast error variances are estimated (M_V is restricted to be diagonal) or that no individual weight can be outside the interval $[0, 1]$.

Differential weighting methods

Differential weighting methods try to minimise the variance of the forecast error based on a covariance matrix. This matrix is usually unknown in practice and has to be estimated through in-sample training and validation sets. Five differential weighting schemes and their empirical comparison are presented in [NG74, WM83].

Least square regression

Least square regression methods try to minimise the sum of the squared error (SSE). This mathematical approach calculates the vector of weights on the past observation and forecast values. However, the squared error of the forecasts is unknown in practice. It has to be estimated through properly selected in-sample training and validation sets.

Outperformance

The outperformance method [Bun75] follows the idea of Bayesian probabilities. One forecast model is considered to outperform another if it offers a smaller absolute forecast error. The weights for the linear combination are determined based on the fractions k out of M executions, one model outperformed the others. Assuming n forecast models, each weight is given by:

$$w = (a_i + s_{i,j})/(\sum_{i=1}^{n} a_i + j) \tag{3.12}$$

where $s_{i,j}$ denotes the number of times method i outperformed all others, and j denotes the number of executions. By setting the assessments a_i ($a_i > 0$), some methods can be preferred a priori over others.

Machine learning algorithms

Although *Adhikari* did not mention machine learning algorithms, they form an important, relatively new class of methods in the field of forecasts combination. Their benefit is that they are able to consider additional input, such as properties of the time series or characteristics of the underlying process. However, literature on these usually non-linear combination models is rather sparse.

Many machine learning algorithms, such as artificial neural networks and support vector machines, are successfully applied to ensemble forecasting [RZS16]. Empirical studies [PL06, STH14, STH15a] show that neural networks outperform well-known linear approaches, such as the simple average, outperformance [dMBT00], or optimal weights [BG69].

Boosting and *bagging* algorithms [AA13] try to obtain smaller forecast errors by combining multiple forecast methods, where each technique has to be at least slightly better than random guessing. Initially, they were designed for ensemble learning in classification tasks, but were later-on adapted to regression problems [AI00]. One famous representative is *AdaBoost* [Sch13] which combines many weak predictors to achieve more accurate forecasts. The output of AdaBoost is a weighted majority vote combining the forecasts of all methods. The weights are derived based on the forecast accuracy of each method on a training set. *Prudêncio* and *Ludermir* [PL04] propose a meta-learning approach for the selection among two candidate models. They applied the J.48 algorithm, respectively a variant of the C4.5 algorithm [Qui93] as meta-learner to decide between the two models.

3.5 Forecast performance measures

The performance of a forecast method is derived by the measurement of its forecast error (or residual). Based on an error measure, the best performing forecast technique for a certain time series can be derived. To evaluate the accuracy of the utilised forecast methods, several metrics are available. A forecast method is considered accurate if the deviations between its forecasts and the actual values are low. For a τ-step forecast into the future for a given time series x, the absolute forecast error is calculated as

$$e_t(\tau) = |x_{t+\tau} - \hat{x}_t(\tau)|, \quad \tau = 1, 2, \ldots \tag{3.13}$$

with $x_{t+\tau}$ being the actual value at the time step $t + \tau$, and $\hat{x}_t(\tau)$ being the forecast for this time step. Naturally, the goal is to minimise $e_t(\tau)$. A variety of measures are employed to evaluate the accuracy of forecasts. Popular forecast accuracy measures are the mean absolute error (MAE), the mean absolute percentage error (MAPE), the root mean square error (RMSE), and the mean absolute scaled error (MASE) [HK06, KW08].

Measures based on percentage errors, such as MAPE, have the advantage of being scale-independent, and can be used to compare forecast methods across different data sets. Scaled error measures remove the scale of the data by comparing the forecasts with the in-sample mean average error of a benchmark forecast method. For instance, the U-statistic by *Theil* [The61] compares the forecasts with the performance of a naive forecast method (it takes the last actual value as forecast).

Another representative of a scaled error measure is MASE. It compares the absolute deviation between the actual values Y_t and the forecasts F_t with the absolute forecast error of the in-sample naive forecast (e.g. the forecast F_t is equal to the last actual value Y_{t-1}). A MASE value below 1 depicts a good forecast accuracy. MASE is calculated as the sample mean of the scaled error q_t over a certain period:

$$MASE = mean(|q_t|). \tag{3.14}$$

The scaled error q_t is defined as

$$q_t = \frac{Y_t - F_t}{\frac{1}{n-1}\sum_{i=2}^{n} |Y_i - Y_{i-1}|} \tag{3.15}$$

RMSE and MAE are representatives of scale-dependent measures. They represent not normalised measures, as their results depend on the value range of the data. Therefore, they can not be used when comparing across different data sets with different scales.

3.6 Summary

This chapter summarised the most important concepts of time series forecasting. Furthermore, it provided a literature review in the areas of forecasting of time series and forecast combination. As we have seen, there exists a large number of methods for forecasting of time series. However, the selection and parameter tuning of a method for a specific task is difficult and requires expert knowledge. Furthermore, all methods have their advantages and disadvantages and there is no

method which performs better than all others for any type of time series [Lem10]. Therefore, a special focus was given to ensemble forecasting, a promising approach that deals with the combination of several forecasting techniques. Thereby, the strengths of multiple forecast methods can be combined. Consequently, we will pursue this approach in the remainder of this work.

In chapter 5, this concept is applied to forecasting of univariate time series by using a machine learning technique as combination strategy. The introduced guidelines will help selecting good ensembles. We will see that choosing only one or two individual methods is more risky than choosing a combination of several methods.

Chapter 4

Learning classifier systems

Learning classifier systems (LCS) resemble genetic, rule-based machine learning techniques that combine ideas from evolutionary computing, reinforcement learning, supervised or unsupervised learning, and heuristics [BK05]. The rule-base or population consists of a set of situation-action mappings (called classifiers), whereas the action can be executed in case the situation matches the current input. An evolutionary algorithm evolves these classifiers, in order to explore the problem space for the best solution for a given task or function. The existing rules are rated for their influence on the system. Classifiers that have a positive effect on the environment tend to be reproduced whereas inaccurate classifiers are eventually deleted. Therefore, the rules that have shown to achieve good results have a higher possibility to be chosen again in later executions. Consequently, the system seeks to maximise the received rewards.

In 1995, *Wilson* and *Butz* [Wil95] proposed the *extended classifier system* (XCS) [BW03]. It combines the principles of learning classifier systems (LCS) with ideas from temporal difference learning. XCS tries to evolve accurate and maximally general rules that cover the state-action space of the underlying problem. The quality of each classifier is based on the accuracy of its prediction. It is therefore also called accuracy-based LCS. XCS gained a majority of attention in the broader research field of Michigan-style LCS (designed for online learning). The characteristic of Michigan-style LCS is that they evolve a single population where the classifiers compete against each other. The benefit of online learning algorithms lies in their ability to continuously adapt their model to changing environments and processes. This is especially important for systems in which the recommended actions are directly executed without human control.

For a more detailed view on XCS and all relevant algorithmic parts and parameters, the reader is referred to [Wil95, Wil03, BW03]. Furthermore, several researchers proposed modifications and additional features [LLWG07, BLW08, SER+16].

The following chapter gives an introduction to two XCS derivatives, XCSR and XCSF, as they are used later-on in the remainder of this thesis. Chapter 5 applies XCSF to ensemble forecasting of time series and chapter 8 presents how XCSR can be applied to congestion detection.

4.1 Extended classifier system for real-valued inputs

In the following, the XCS for real-valued inputs, *XCSR* [Wil00], is introduced. XCSR is an online learning classifier system that learns non-linear, real-vectored classification tasks. Its advantage over the traditional XCS is its ability to take real-valued input that is usually found in real-world

environments. XCS has been successfully applied in a variety of real-world applications [Bul04], such as data mining, robotics, and traffic control [PBS+09, BST+04]. Besides its knowledge base, XCSR consists of three main components: the performance component, the discovery component, and the reinforcement component.

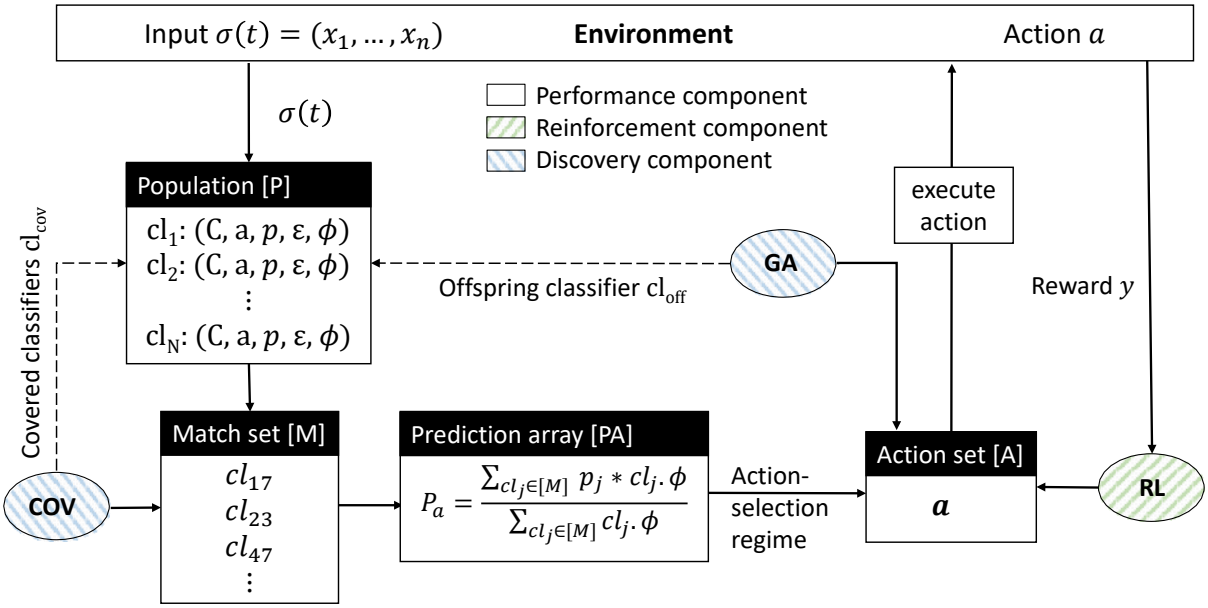


Figure 4.1: Schematic illustration of the standard XCSR. GA=Genetic Algorithm, COV=Covering, RL=Reinforcement learning

4.1.1 Knowledge representation

XCSR stores its knowledge in form of a set of classifiers. Each classifier comprises a couple of attributes:

1. The **condition** C determines a certain subspace of the problem space by encoding a geometric structure (such as rectangles or ellipsoids) (Figure 4.2). It therefore denotes the part of the problem domain where the classifier is applicable.
2. The **action** a defines a reaction executed on the environment in case C matches the input.
3. The **experience** exp denotes how often the classifier was part of an action set.
4. The **numerosity** num shows the number of macro-classifiers represented by this classifier.
5. The **prediction** p estimates the average future payoff of the executed action.
6. The **prediction error** ϵ estimates the mean absolute deviation between the prediction and the actual outcome.
7. The **fitness value** ϕ can be roughly interpreted as an inverse of ϵ. It provides a measure for the relative accuracy of the classifier.

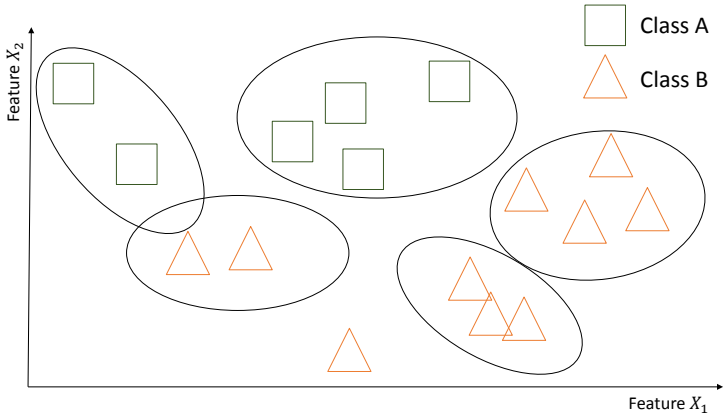


Figure 4.2: Schematic illustration of a two-dimensional feature space with two classes A and B. The conditions of XCSR's classifiers are illustrated by ellipsoids.

4.1.2 Performance component

The performance component determines which action is to be executed in the environment in response to the given input. Figure 4.1 depicts the standard XCSR and its action-selection process. Initially, XCSR is provided with an external input from its environment. Typically, the input contains a vector of sensory values which is called feature vector $\sigma(t) = (\sigma_1, \ldots, \sigma_n)$. The population $[P]$ is scanned for classifiers matching the input $\sigma(t)$ contributing to the match set $[M]$. Typically, the classifiers in the match set represent different actions. Therefore, an action-selection process has to be executed. For each action a represented in $[M]$, a fitness-weighted average, the system prediction P_a, is computed as

$$P_a = \frac{\sum_{cl \in [M]} cl_j.\phi * p_j}{\sum_{cl \in [M]} cl_j.\phi)}. \tag{4.1}$$

Note that the term 'system prediction' originates from the XCS terminology and that we differentiate this from the term 'forecast' that is used to denote a calculated time series value at time $t + k, k > 0$. The P_a values are then added to the prediction array $[PA]$. Some actions may not get any value since they are not represented in the match set. Afterwards, an action is chosen from $[PA]$ based on the selected action-selection regime. The selection can either be random (*Exploration*), probabilistic, or deterministic (*Exploitation*). Exploitation means that the action is chosen in a way that the classifier's payoff is maximised. Finally, the action set $[A]$ consist of the subset of classifiers of $[M]$ having the chosen action a which is finally executed. Based on the influence of the executed action on the environment, the classifiers from the action set are updated. The update process is executed by the reinforcement component.

4.1.3 Reinforcement component

After the execution of a, the prediction p, the prediction error ϵ, and the fitness ϕ of each classifier in $[A]$ is updated. Based on their influence on the environment, these classifiers receive a positive or negative reward (*Reinforcement learning*). The reward r is eventually used to refine

the attributes by means of gradient-descent based techniques. Conventionally, the *Widrow-Hoff delta rule* [WH88] is incorporated to accomplish this step. The prediction error ϵ of each classifier cl from the last action set is updated in dependence of the deviation between the prediction $cl.p$ and the actual reward r by

$$cl.\epsilon = cl.\epsilon + \beta(|r - cl.p| - cl.\epsilon) \tag{4.2}$$

and the prediction p is updated by

$$cl.p = cl.p + \beta(r - cl.p). \tag{4.3}$$

The learning rate β defines the speed of the adaptation process. Higher learning rates lead to faster adaptation but result in higher variance.

The fitness of the classifier cl_j is updated in dependence of the relative accuracy of cl_j compared to the accuracy of all other classifiers in the action set. A classifier is assumed maximally accurate, if its prediction error ϵ is below a threshold ϵ_0. Thus, it is assigned a value of $\kappa = 1$. Less accurate classifiers receive an accuracy of $\kappa < 1$ determined by a parametrisable power function. The relative accuracy of classifier j is calculated as

$$\kappa_j^{'} = \begin{cases} 1, & cl_j.\epsilon < \epsilon_0 \\ \frac{\kappa_j}{\sum_{i \in [A]} \kappa_i}, & \text{otherwise.} \end{cases} \tag{4.4}$$

and its fitness is updated by

$$cl_j.\phi = cl_j.\phi + \beta(\kappa_j^{'} - cl_j.\phi). \tag{4.5}$$

The update process of other parameters is not further described here, but can be looked-up in [BW03].

4.1.4 Discovery component

As figure 4.1 depicts, there is a further part – the *discovery component*. It is responsible for the creation and deletion of classifiers.

Covering mechanism: Whenever $[P]$ contains no classifier cl_j whose condition C encompasses the current situation $\sigma(t)$ or it only consists of classifiers with insufficient quality, a so-called *covering mechanism* (COV) is activated. This assures that at least one classifier cl_{cov} is generated and guarantees immediate response to any input vector $\sigma(t)$ at any time t_i. A genetic algorithm (GA) creates classifiers with their condition matching $\sigma(t)$ and a randomly chosen action. The prediction p, the prediction error ϵ, and the initial fitness ϕ are initialised according to predefined values. The new classifier is also added to the population.

Genetic algorithm: Besides the covering mechanism, a *steady-state niche genetic algorithm* (GA) comes into operation, to refine the geometric shapes during the learning process. In certain intervals, defined by parameter θ_{GA}, XCSR executes the GA to discover new rules and to explore the problem space. The GA is executed only on classifiers from the match set (Niche GA). It thereby selects two parental classifiers and creates two offspring classifiers by mutation

and crossover operations of the parents. A classifier is mutated by adding or subtracting a random offset to or from its condition representation. The new condition is initialised with random values for c and s and the prediction error and fitness are reset. Finally, the classifier is inserted into the population.

Subsumption deletion: An offspring classifier cl_{off} can be subsumed by another classifier in $[M]$ that additionally encompasses the condition of the offspring classifier entirely. The more general classifier has to have sufficient experience, and a prediction error ϵ smaller than the target error ϵ_0. In that case, cl_{off} is not added to the rule base since it does not encode new knowledge, but the numerosity of the subsuming classifier is increased by one.

4.2 Extended classifier system for function approximation

The extended classifier system was initially utilised for the task of function approximation in 2002 [Wil03]. In his article, Wilson presents an *XCS for function approximation* (XCSF) that aims at approximating non-linear, multi-dimensional functions $y = f(\sigma(t))$ where $\sigma(t) \in \Re^n$ is the input and y is the function value prediction. XCSF combines *gradient-based* local learning [AMS97] and a *steady-state niche genetic algorithm* (GA) [BLW08]. More precisely, XCSF attempts to partition the problem space into several subspaces and learns linear approximations of the targeted function f within the corresponding niches.

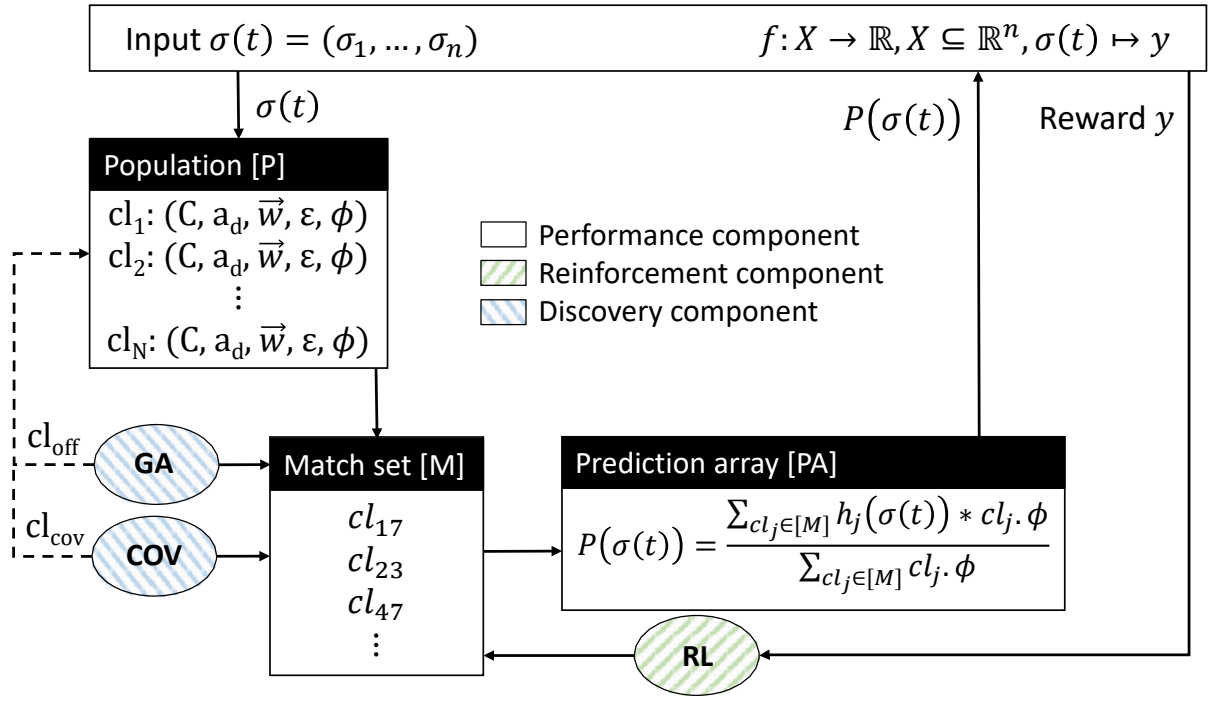


Figure 4.3: Schematic illustration of the standard XCSF. GA=Genetic Algorithm, COV=Covering, RL=Reinforcement learning

As can be seen in figure 4.4, XCSF approximates the underlying problem space in a piece-wise linear fashion. The rectangles exemplarily represent the conditions of different classifiers encoding different subspaces. However, other shapes, such as hyper-ellipsoids [But06], have been proposed. Classifiers can encode overlapping subspaces and there might also be parts of the function which are not (yet) covered by any classifier (knowledge gaps).

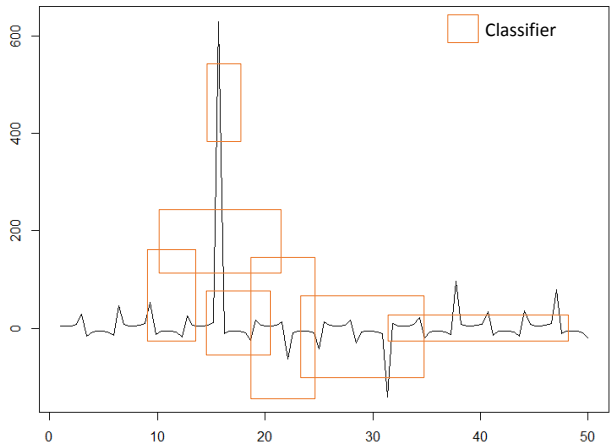


Figure 4.4: Schematic illustration of the piece-wise approximation of a two-dimensional function by XCSF. Rectangles represent the classifiers' conditions.

4.2.1 Knowledge representation

A single classifier of XCSF comprises the following attributes:

1. The **condition** C that determines a certain subspace of the problem space by encoding a geometric structure (for example rectangles or ellipsoids [But06]),
2. an **action** a that defines a reaction to be executed on the environment in case C matches the input (in the case of XCSF, only a single dummy action a_d is used),
3. a **weight vector** $\vec{w} = (w_0, w_1, \ldots, w_n)$ that represents the coefficients for the linear approximation with w_0 being an offset weight,
4. an **error estimate** ϵ that reflects the mean absolute prediction error of the linear approximation,
5. a **fitness value** ϕ that can be roughly interpreted as an inverse of ϵ, which represents the relative predictive accuracy of the linear approximation.

In contrast to other XCS variants, XCSF does not store the prediction as a scalar, but rather calculates the prediction in dependence of the input vector $\sigma(t)$. Furthermore, XCSF does not output an action, but rather a predicted function value. Therefore, the concept of an action set is dropped. Besides these changes, XCSF works more or less similar to XCSR. The focus of the following sections lies on the differences compared to XCSR.

4.2.2 Performance component

As illustrated in figure 4.3, a single iteration through XCSF's main loop works as follows. At each time step t, XCSF retrieves an n-dimensional situation/feature vector $\sigma(t)$ from its environment. Next, the population $[P]$ is scanned for classifiers which match the given feature vector. These classifiers in turn constitute the match set $[M]$ which are used to determine the function value prediction.

XCSF combines overlapping subspaces represented by classifiers $cl_j \in [M]$ (i.e. *environmental niche*) according to their fitness value $cl_j.\phi$. This combination, more precisely the system prediction $P(\sigma(t))$, is used as output value, estimating the function value at point $\sigma(t)$. It is determined by means of a fitness-weighted sum of all linear approximations at the problem space's site $\sigma(t)$ by

$$P(\sigma(t)) = \frac{\sum_{cl\in[M]} cl_j.\phi * h_j(\sigma(t))}{\sum_{cl\in[M]} cl_j.\phi} \tag{4.6}$$

where the linear approximation of a classifier cl_j is calculated as

$$h_j(\sigma(t)) = w_0 + \sum_{i=1}^{n} w_i \cdot \sigma_i. \tag{4.7}$$

Therefore, more accurate classifiers contribute stronger to the final prediction.

4.2.3 Reinforcement component

The refinement of the matching classifiers is the task of XCSF's reinforcement component. The reward r is eventually used to refine the attributes ϵ and ϕ of each classifier $cl_j \in [M]$ by means of gradient-descent based techniques. The process is equivalent to XCSR. For the adaptation of the coefficients $\vec{w}$ of the linear approximation, the *recursive least squares* (RLS) algorithm [BLW08] is used. Given the input vector $\sigma(t)$ and the target function value y, the weight vector $\vec{w}$ is updated by

$$\vec{w} = \vec{w} + \vec{k}[y - (\sigma(t) - \vec{m})\vec{w}]. \tag{4.8}$$

Parameter $\vec{m}$ denotes the centre of the ellipsoid or rectangle and $\vec{k}$ is the gain vector. For further details on the update process, the reader is referred to *Butz et al.* [BLW08].

4.2.4 Discovery component

During covering, the attributes of the newly created classifier cl_{cov} are initialised. The condition C (representing an hyper-ellipsoid) is set according to the current input $\sigma(t)$. The condition's centre is set equal to $\sigma(t)$ whereas the stretch and angle are randomly initialised. The offset weight of the weight vector $\vec{w}$ is set to the target function value whereas the other weights are initialised as 0. Other classifier attributes, such as the fitness and the prediction error, are initialised to pre-defined values. Other mechanisms of the discovery component work analogously to XCSR.

4.3 Summary

In this chapter, the most important concepts and representatives of extended classifier systems were presented. These rule-based machine learning techniques use genetic algorithms to evolve new rules at runtime. Over the years, the general structure was extended and adapted by several researchers for specialised tasks. The XCSR is an adaptation of XCS to deal with real-valued inputs, and the XCSF is a specialised version for function approximation. The benefit of XCS is its ability to gain new knowledge and to memorise previous actions and outcomes at runtime. This allows XCS to evolve and improve its rules at runtime.

In later chapters, we will see how these concepts can be applied to diverse tasks, such as combining several forecast methods with XCSF, or detecting congestion on highways and in cities with XCSR.

Part II

Contributions

Chapter 5

Learning and self-adaptive forecasting of time series

The main purpose of adding adaptation and self-optimisation capabilities to technical systems in general, is to allow for resilient and flexible solutions. Current research activities focus on turning autonomic computing [KC03] and organic computing [MS04] systems from robust into resilient systems. Thus, the control mechanisms encapsulating the self-adaptive and self-organising capabilities do not only react to detected disturbances and dissatisfying system performance, but are trying to foresee upcoming problems. For the pure control tasks, this means to make forecasts that serve as additional input for the self-adaptation mechanism defined by the architecture. In particular, the system proactively changes the control strategies to predicted future states, also trying to become aware of the impact of the decisions taken. Time series forecasting has been used to support the decision-making process, and to mitigate uncertainty of future trends in real-world domains [KK12, MJG90, VSSB13, STH16b]. Furthermore, combinations of multiple forecasts from several independent forecast methods provide more accurate forecasts then when relying only on one single forecast technique [AA14]. Consequently, the combination of different forecast techniques reduces the model selection risk.

Typically, the forecasts of the different models do not contribute equally to the combined forecast, but are weighted according to their expected accuracy. The performance of the combined forecast depends highly on the chosen weights. One possible approach to configure these weights appropriately is to find the best configuration offline, i.e. at design-time. Unfortunately, this approach is very time-consuming and determines settings that are only optimal in the sense of working best on average for all considered situations. Since there is no best forecast technique, there is also no best set of weights for all situations. The optimal weights are different for each time series, thereby a static assignment of weights will always be suboptimal. Therefore, the assigned weights have to be adjusted dynamically at runtime, depending on the currently observed time series pattern and the history of the current situation. Since a variety of different techniques exists to realise the forecasting tasks, the question arises which of them is the best possible choice in general or in each particular situation. Hence, the possibility to let the system automatically learn a situation-to-forecast-technique mapping, in order to decrease the deviation between forecast and actual value over time, is investigated.

The problem of forecasting with univariate time series and especially multi-model ensemble forecasting [Arm01] is treated as a machine learning problem. Self-adaptive machine learning techniques have the ability to learn and improve their knowledge at runtime. Machine learning techniques are used as a meta-learning approach for the weighted combination of a given set of

forecast methods. A highly self-adaptive, rule-based machine learning technique, the *extended classifier system for function approximation* (XCSF), is adapted to multi-model ensemble forecasting. XCSF learns the best combination during runtime in a self-adaptive fashion. Furthermore, it evolves its knowledge by the principle of reinforcement learning. Therefore, no system expert is needed to determine the best combination of the applied forecasting techniques at design time. Thorough time series analysis by humans is often not feasible in practical applications. XCSF relieves an engineer from complex design-time-situated tasks related to forecast selection and forecast combination.

The remainder of this chapter is structured as follows: First, the formal concept of forecast combination is mapped to machine learning problems. Based on this theoretical concept, it is presented how an an automatic approach, in particular the learning classifier systems XCSF, can be used to tackle this task. On the basis of differently characterised univariate time series, forecast combination by means of XCSF is compared to alternative combination approaches.

5.1 Problem formulation

> Nobody wants a prediction that the future will be more or less like the present, even if that is, statistically speaking, an excellent prediction.
>
> Nathan Myhrvold

The combination of forecasts from different methods is an effective way to improve the forecast accuracy. In general, the forecast combination problem can be expressed as finding a vector of the optimal weights for an input of forecasts from different forecasting models, to obtain the optimal combination of these forecasts. In contrast to static approaches, such as the widely used simple average, machine learning algorithms are able to adapt the weights of the linear combination in accordance to past forecast errors. The combination problem is considered a learning problem, where a machine learning technique tries to infer the possibly non-linear dependence between the input (set of individual forecasts) and the output (combined forecast). The goal of the combination method (the so-called meta-learner) is to automatically learn a function g:

$$g(sit) \rightarrow (w_1, \ldots, w_n). \tag{5.1}$$

The term sit defines the observed situation and w_1 to w_n (with $n > 1$ and $w_1 + \ldots + w_n = 1$) resemble the weights for the individual forecast techniques (the so-called base learners). Within each evaluation cycle of the system, the weights of the forecast ensemble are altered according to the estimated reward of g. The reward is defined by the accuracy of the combined forecast in dependence of the actual time series value. Usually, the accuracy is defined by the absolute forecast error, but any other forecast error measure can be used as well. In general, g is a function that maps situations to the configuration of the weights. This implies two basic tasks:

1. A description of the situation is needed that serves as parameter for g, and

2. a combination technique is needed that is able to improve this mapping over time with increasing experience.

As the aim is to combine several forecasts, the intuitive solution is to build the input vector sit consisting of these forecasts. Additional data, such as the last n time series values or more complex measures describing the characteristics of the time series, can be added (i.e. seasonality, serial correlation, self-similarity etc. [WSH06]). However, a larger input vector leads to a larger search space which might only complicate the underlying problem without contributing additional information. Furthermore, more input parameters lead to a higher-dimensional function to be approximated. In general, the concrete realisation of the situation description depends on the respective system. For the purpose of this work, an approach that is as close to the standard methods as possible is used. Therefore, the first approach is utilised and the term sit is defined as the vector of the individual forecasts.

For the second task, a number of techniques have been proposed that consider historic measurements to find the optimal vector of weights for a given set of forecast methods (see section 3.4). However, static combination strategies are not able to encode the time-varying aspects of the problem. In contrast, adaptive models determine the optimal weights at runtime, mostly by considering the recent forecasts accuracies of the forecast methods in the ensemble. The weights are updated iteratively in accordance to the individual performances of the base predictors. Suitable representatives for this problem derive from the machine learning domain. From the machine learning point of view, each individual forecast method can be seen as a base learner trying to deduce the underlying model of a given time series. Based on his internal model, each learner gives his independent vote F_i (or forecast in terms of time series forecasting). The output of each individual forecast technique is considered with varying impact, weighted and combined by a meta-learner, according to its expected accuracy. The problem of finding the optimal weights can be formulated as a minimisation problem:

$$\underset{\vec{w}}{\operatorname{argmin}}(|X(t) - \hat{F}(t)|) \tag{5.2}$$

with $X(t)$ being the actual value at time step t, and $\hat{F}(t)$ being the combination of forecasts for time step t, weighted by $\vec{w} := (w_1, \ldots, w_n)$. The optimisation criterion is the minimisation of the forecast error. This can be achieved by exploring the search space of all possible weight combinations. The key problem is that these weights are not static among a certain set of forecast methods, but are dependent on the current situation within the time series, and the time series itself.

In the following, a new approach is introduced, that applies a representative of the class of *learning classifier systems* (LCS) as meta-learner to find the optimal combination of forecasts. An LCS evolves rules which encode situation-to-action mappings. A situation in equation 5.1 represents the condition part in the learning function, while the particular weights for combining the forecasts of the available techniques define the action part. The following gives a brief introduction to XCSF [BLW08]. We move on, applying this evolutionary, rule-based machine learning technique to evolve rules at runtime that encode the optimal weights for the combination of forecasts from several forecast techniques.

5.2 Multi-model ensemble weighting with XCSF

In the following, the idea of utilising XCSF as a forecast combination method is presented. In analogy to the concept of stacking (see section 3.4.1), we apply XCSF as a meta-learner. First, some formalisms and interpretations have to be introduced.

Representation of the input
As stated in section 3.1, a time series can be formalised as $ts = X_1, \ldots, X_n$, where n determines the length of ts. The time series can be interpreted as a function, where the sampling of the domain is determined by the time a certain value appears. Thus, instead of a uniform sampling, the time series is, conceptually spoken, ordered by time. In each learning step, XCSF is presented the values $\hat{F}_t$ of multiple forecast methods for the future time step $t + m$. Forecasting strives to look ahead the current value of ts, and to predict the value X_{t+m}, m steps in the future. Without loss of generality, the following experiments focus on forecasting exactly one step in the future. Thus, $\hat{F}_t$ is defined as a combined forecast for the future value X_{t+1}. With XCSF as ensemble time series forecast combiner, F_i for $i = 1, \ldots, k$ is further defined as the calculated value of the i-th forecast method. Accordingly, the situation vector $\sigma(t)$ that is retrieved by XCSF is now defined by $\sigma(t) = (F_1, \ldots, F_n)$, i.e. XCSF is presented the individual forecast values of each method in the ensemble. Accordingly, XCSF has to approximate the underlying n-dimensional function.

Representation of the classifiers
To encode the condition part of a classifier, rotating hyper-ellipsoids are used as introduced by *Butz et al.* in [But06]). Ellipsoids are able to cover the state space better than rectangles, with less classifiers needed. The location and shape of the ellipsoid is defined by a centre point, its stretch defined by its radii, and a rotation angle (Figure 5.1). A classifier condition is now defined by a vector $\vec{m}$ denoting the centre of the hyper-ellipsoid (in case of more than two dimensions) and a transformation matrix σ. The condition is defined as $C = (F_1, \ldots, F_n)$ where $F_1, \ldots, F_n$ are the forecasts of the individual base learners. The condition matches a point if it lies within a radius of one from its centre.

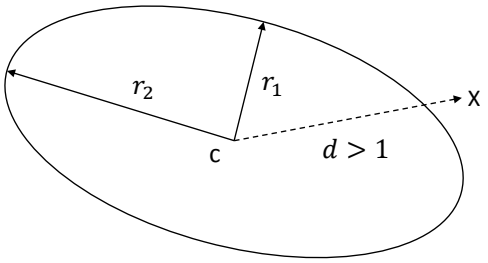


Figure 5.1: Illustration of a 2D ellipsoid with its centre point c and its radii r_1 and r_2. The point X lies outside the ellipsoid since the distance d is greater than one.

Linear approximation
For the combination of n forecast methods with their forecasts F_1 to F_n and their according

coefficients $\vec{w} = \{w_0, w_1, ...w_n\}$, the fitness-weighted linear approximation (here: the combined forecast value) is calculated as:

$$P(\sigma(t)) = \frac{\sum_{cl_j \in [M]} h_j(\sigma(t)) * cl_{j.\phi}}{\sum_{cl_j \in [M]} cl_{j.\phi}} \tag{5.3}$$

whereas the linear prediction $h(\sigma(t))$ is calculated by

$$h(\sigma(t)) = w_0 + w_1 * F_1 + \ldots + w_n * F_n. \tag{5.4}$$

This leads to an approximated linear function $h(\sigma(t))$ for the region covered by the classifiers in $[M]$ that intersects the ordinate approximately at the actual value's height in the problem function. Naturally, the proximity to the actual value X_{t+1} also depends on the quality of the selected ensemble of forecast methods.

Covering mechanism
In the case of covering, $\vec{w}$ is usually randomly initialised with weights from $[-1.0, 1.0]$. To provide XCSF with a sufficient initial prediction, the offset weight w_0 of the initialised weight vector $\vec{w}$ for the newly created classifier cl_{cov} is set to

$$w_0 = \frac{\sum_{i=1}^{n} F_i}{n} \tag{5.5}$$

which resembles the mean of the multiple forecasts delivered in the situation vector $\sigma(t)$.

Reward and reinforcement
After forecasting the value for time step $t+1$ ($\hat{F}_t$), the reward r_{imm} that is retrieved by XCSF is set to the actual time series value X_{t+1}. Accordingly, the absolute error is calculated by $absError = |\hat{F}_t - X_{t+1}|$ and further incorporated to update the classifier attributes of all $cl_i \in [M]$ using recursive least square. Remember that XCSF's overall objective is to maximise the retrieved reward. In this context, this objective is equivalent to reducing the absolute forecast error which eventually results in more accurate forecasts.

5.3 Evaluation of individual methods and ensemble combination strategies

The capabilities of XCSF for multi-model ensemble forecasting of univariate time series are demonstrated with several time series from different real-world domains exhibiting different characteristics. In particular, the following experiments address these questions:

- Which combination strategy reduces the forecast error the most?
- How does the number of combined forecasts influence the performance?
- Is XCSF able to learn the underlying task by creating general and accurate classifiers?

5.3.1 Experimental setup

The experimental process works as follows. First, each time series is normalised to the value range of $[0; 1]$. XCSF assumes that each value of the input vector is within this value range. Second, for every time step of each time series, a one-step forecast is made. Third, based on these forecasts and several forecast error measures, the forecast errors and several combination strategies are compared to our XCSF approach. The multi-model combination approach using XCSF is compared against the performance of several individual forecast methods and against three well-known linear combination strategies.

Time series

To evaluate our combination strategy, we use ten time series from real-world domains. These daily data sets are from the Quandl Data Library[1], and the FRED economic data library[2]. Their time plots can be seen in figure 5.2. Further description is given in table 5.1. These time series exhibit different characteristics, such as trends, seasonal patterns, or non-stationary behaviour. Due to their noisy, non-stationary, and sometimes chaotic behaviour, economic time series are considered especially difficult to forecast.

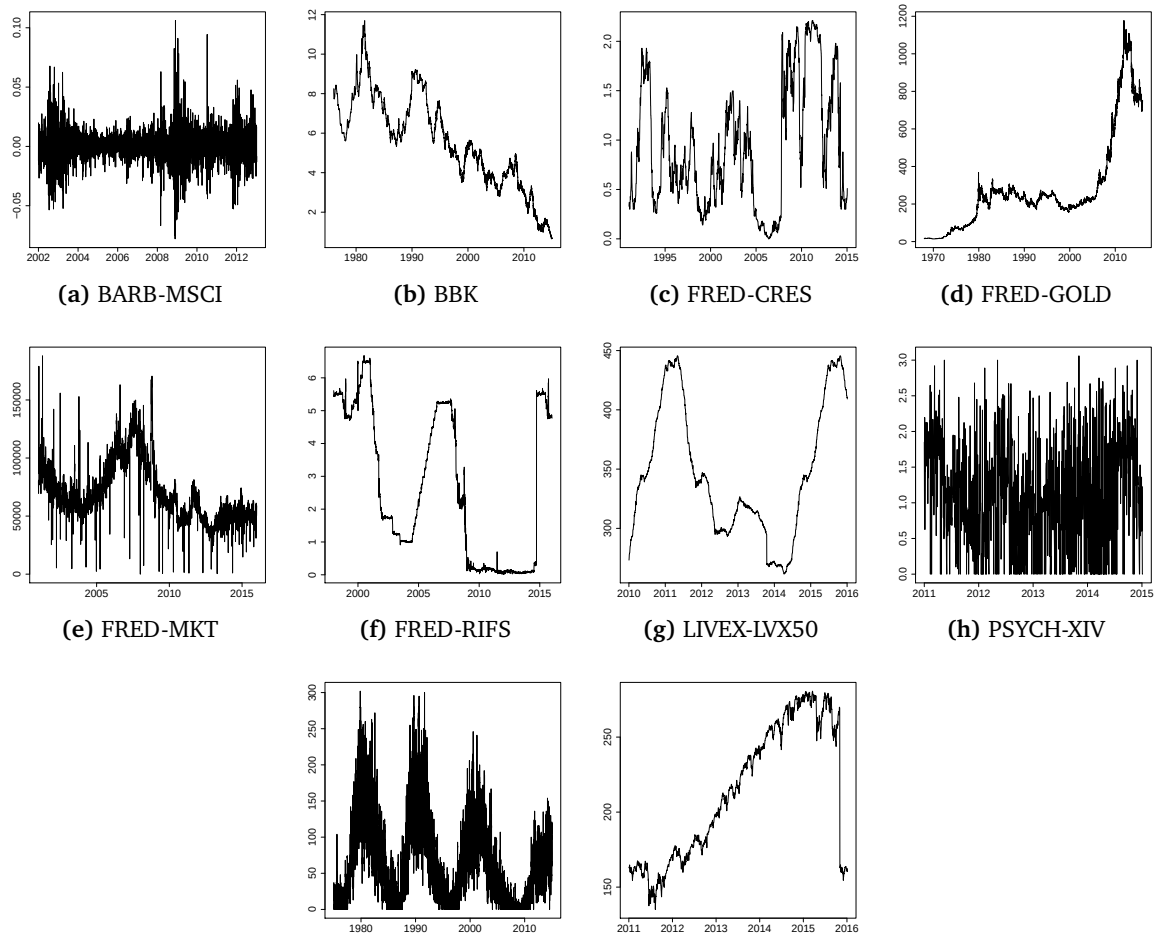
Table 5.1: Description of the time series data sets used for evaluation.

Time series	Description	Type	Length
BARB-MSCI	MSCI Equity Index	stationary, non-seasonal	2,918
BBK	Bundesbank: Yields on debt securities outstanding issued by residents	non-stationary, trend, seasonal	9,925
FRED-CRES	Cleveland Financial Stress Index	non-stationary, seasonal	5,810
FRED-GOLD	LBMA Gold Price: Daily Prices	non-stationary, trend, non-seasonal	12,118
FRED-MKT	Total commercial paper issues with a maturity Between 1&4 days	stationary, non-seasonal	3,813
FRED-RIFS	15-Day AA financial commercial paper interest rate	non-stationary, seasonal	4,183
LIVEX-LVX50	Daily prices of 50 Fine Wine Index	non-stationary, seasonal	1,633
PSYCH-XIV	Sentiment volume ratios for stocks	stationary, non-seasonal	921
SUNSPOTS	Daily total number of sunspots	stationary, seasonal	14,761
SPDJ-SPFTR	S&P Dow Jones 500 Futures Index	non-stationary, trend, seasonal	1281

Individual forecast methods

The ensembles used for the evaluation consist of subsets of the following forecasting methods provided by the forecast package for `R` [HK08]. These particular base predictors are chosen

[1] http://www.quandl.com
[2] https://research.stlouisfed.org/fred2/



(a) BARB-MSCI **(b)** BBK **(c)** FRED-CRES **(d)** FRED-GOLD

(e) FRED-MKT **(f)** FRED-RIFS **(g)** LIVEX-LVX50 **(h)** PSYCH-XIV

(i) SUNSPOTS **(j)** SPDJ-SPFTR

Figure 5.2: Time plots showing the ten time series with different length and characteristics used for the evaluation.

because of their structural diversity (complexity, parameters, and architecture). Furthermore, simple approaches are often surprisingly robust. Thereby, we follow the guidelines for the combination process as presented in section 3.4.2. This selection allows us to evaluate several heterogeneous ensembles with different numbers of methods.

Moving average (MA) calculates the one-step forecast based on the recent time series values. Its equal to an ARIMA(0,0,1) model. It has the advantage of quick low cost updates and accurate short-term forecasts. Its disadvantage is that it is not coping well with trend or seasonality.

Cubic spline's smoothing (CS) model is equivalent to an ARIMA(0,2,2) model, but with a restricted parameter space. The advantage of cubic spline over the full ARIMA model is that it provides a smooth historical trend, as well as a linear forecast function.

ARIMA(1,0,1) with drift is a first-order autoregressive model with one order of a moving average process. The linear drift term resembles a linear regression with fitted ARIMA errors.

The **exponential smoothing state space model (ETS)** is fully automatic and estimates the model based on the given time series only. Based on measures for error, trend, and seasonality, ETS automatically chooses one of 30 different smoothing models and applies it to forecast the given time series.

The **random walk with drift** model (RW) is calculated as

$$Y_t = c + Y_{t-1} + Z_t \tag{5.6}$$

where Z_t is a normal error.

The **simple exponential smoothing model (ES)** equals ARIMA(0,1,1) and the **double exponential smoothing** (DES) equals ARIMA(0,2,2). More information on these models is given in section 3.3.2.

Based on a brief parameter study by grid-searching on a validation set, the input of CS, ES, and DES is defined as the last ten time series values. ETS and ARIMA make their forecasts based on the last 25 observations. The MA method considers the last three values. The random walk method estimates the drift based on the last ten observations.

Reference combination strategies

The implementation of XCSF makes use of the *XCSF-Ellipsoids Java* project [SB08] that was made available by *Stalph* and *Butz* in 2008. As reference solutions, several combination strategies are selected that belong to different classes as presented in section 3.4.3.

The **simple average (SA)** is a statistical method, representing the most widely used forecast combining technique. It is easy to understand and to implement. Surprisingly, it often outperforms more complex strategies [Lem10]. However, it does not respect the individual performances of the contributing algorithms by assigning equal weights to each method and is sensitive to extreme errors.

The error-based **optimal weights method (OW)** [BG69] estimates the linear weights to minimise the error variance of the combination (assuming unbiasedness for each individual forecast). The sum of the resulting weights equals one and no individual weight can be outside the interval $[0, 1]$. The weights are estimated based on a short history of n absolute forecast errors for each forecast model. A brief parameter study with $n = \{5, 10, 15, 20, 25\}$ indicated that $n = 5$ offers the best results.

Outperformance (OP) [dMBT00] represents a robust non-parametric method. It calculates each individual weight as the probability that its respective forecast will perform best (in the smallest absolute error sense) on the next iteration. Each weight is estimated as the fraction of occurrences in which its respective forecasting model has performed best in the latest k executions. A brief parameter study for $k = \{5, 10, 15, 20, 25\}$ is conducted. The best performance is achieved with $k = 20$. Therefore, this parameter setting is used in the following evaluation. By setting the a priori values a_i as unity, the initial weights are equal for all methods.

Performance measures

Many different forecast accuracy measures have been proposed. *Hyndman* and *Koehler* [HK06] provide an overview and compared different measures. Often, accuracy measures are needed that allow to compare several forecasting methods across different data sets. The following error measures help us detecting the relative strengths and weaknesses of the evaluated combination strategies. To compare several forecasting methods across data sets with different scales, the following scale-independent error measures are used.

U-statistic: Theil's U-statistic is a relative accuracy measure, comparing the RMSE of the proposed method against the RMSE of a naive one-step ahead forecast. It is calculated as

$$U = \sqrt{\frac{\sum_{t=1}^{n-1} (\frac{F_{t+1}-Y_{t+1}}{Y_t})^2}{\sum_{t=1}^{n-1} (\frac{Y_{t+1}-Y_t}{Y_t})^2}} \tag{5.7}$$

where Y_t is the actual value of a point for a given time period t, n is the number of data points, and F_t is the forecast. If $U(\tau) < 1$, the forecasts deliver a better result than the naive forecasts. For $U(\tau) = 1$ the forecasts are seen to be equally good as the naive forecasts. This measure equals zero if the predicted and the actual values are equal, therefore showing optimal forecasts over the whole forecast horizon.

SMAPE: The symmetric mean absolute percentage error (SMAPE) is an accuracy measure based on percental (or relative) errors. SMAPE has a lower bound of 0% and an upper bound of 100%. It is calculated as follows:

$$SMAPE = \frac{1}{n} \sum_{t=1}^{n} \frac{|F_t - Y_t|}{|F_t| + |Y_t|} * 100. \tag{5.8}$$

MASE: The mean absolute scaled error (MASE) [HK06] compares the forecast accuracy of the proposed model with the average forecast error of a naive one-step forecast:

$$MASE = \frac{\sum_{t=1}^{n} |Y_t - F_t|}{\frac{n}{n-1} \sum_{i=2}^{n} |Y_i - Y_{i-1}|}. \tag{5.9}$$

Its lower bound is 0. The method with the lowest MASE offers the best accuracy.

Parametrisation

The efficiency of the XCSF's learning process depends on a number of learning parameters (Table 5.2). Within this evaluation, XCSF is mostly parametrised with the default values as suggested by *Butz* and *Wilson* [BW03, Wil03].

The N parameter determining the *maximum number of classifiers* in the population $[P]$ has to be chosen in dependence of the problem size. Initially, XCSF starts with an empty population. For ensembles of two, three, and five forecast methods, the population consist of a maximum of

Table 5.2: Parameter settings of XCSF.

Learning parameter	Assigned value	Default value
Rule base size limit N	$200/300$	problem dependent
Learning rate β	0.1	$0.1 - 0.2$
Initial fitness ϕ_{ini}	0.01	0.01
Accuracy determination	$\alpha = 1$, $\nu = 5$	$\alpha = 1$, $\nu = 5$
Error threshold ϵ_{ini}	0.01	1% of value range
GA activation interval θ_{GA}	50	$25 - 50$
GA crossover probability χ	1.0	$0.5 - 1.0$
GA mutation probability μ	0.05	$0.01 - 0.05$
Classifier deletion thresholds	$\theta_{del} = 20$, $\delta = 0.1$	$\theta_{del} = 20$, $\delta = 0.1$
Classifier subsumption θ_{sub}	20	20
Initial condition spread r_0	0.5	1
Recursive least squares	$\delta_{RLS} = 1000$, $\lambda = 1$	$\delta_{RLS} = 1000$, $\lambda = 1$
Error reduction factors	$\phi_{red} = 0.1$, $\epsilon_{red} = 1.0$	$\phi_{red} = 0.1$, $\epsilon_{red} = 1.0$

200 classifiers. To account for the bigger search space with a set size of seven forecast methods, the maximal population size is set to 300. In case of reaching this limit, the least experienced classifiers are removed (with respect to the classifier deletion thresholds).

Within the GA, *subsumption* of offspring classifiers by their parents is used, as highly similar classifiers are unlikely to encode additional knowledge. The usage of the *compaction* and *condensation* mechanisms is omitted. They aim at reducing the number of redundant classifiers within the population. *Tournament selection* is used to select parents on a fitness-proportionate basis for crossover within the GA. Each classifier's condition is represented as a rotating ellipsoid [BLW08] which is assumed to cover the state space better than rectangles. Accordingly, the recursive least squares approximation was used for parameter estimation of the prediction values of the classifiers.

Again, for the sake of brevity, it is noted that a certain degree of familiarity with the standard XCS(F) is assumed. For a more detailed view on XCS(F) and all relevant algorithmic parts and parameters, the interested reader is referred to [Wil03, BW03, LLWG07, BLW08].

5.3.2 Experimental results: Two forecast methods

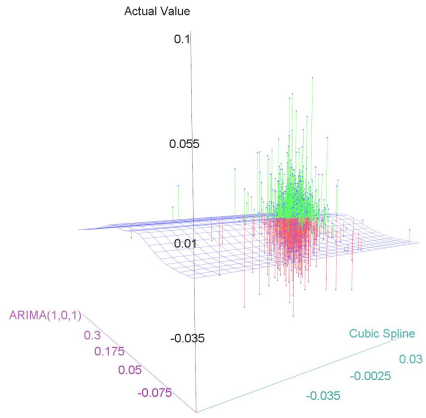
The following section presents the results of the evaluation of an ensemble consisting of two forecast methods, ARIMA(1,0,1) and cubic spline. These two methods are chosen as they use different model representations. However, ARIMA is considered to provide better forecasts than cubic spline. Therefore, the experiments will show that the combination strategies assign higher weights to ARIMA. Initially, all strategies assume equal weights.

Depiction of the combination problem functions

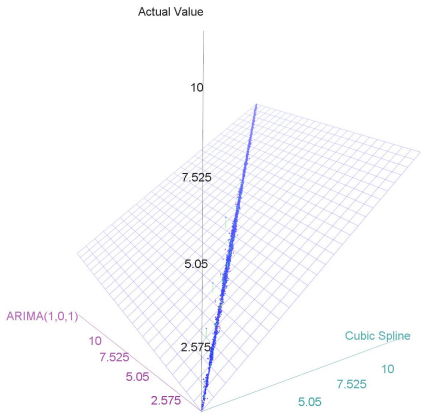
To provide the reader with an idea of how the problem function which XCSF has to approximate may look like, the scatter plots in figure 5.3 depict fitted regression surfaces for all time series. This visualisation is a statistical process for estimating the relationships among variables. The surface is determined by the forecast values F_1 and F_2 derived by the two forecast methods ARIMA(1,0,1) and cubic spline, as well as the time series value X_{t+1}. By approximating such a function, XCSF learns implicitly how much influence each feature in $\sigma(t)$ (i.e. $F_1, \ldots, F_k$) has related to each other, when exactly these values are to be forecasted. Most of the depicted regression surfaces are characterised by a cluster along the main diagonal. This observation is attributed to the fact that the individual forecasts are already rather precise. Therefore, this cluster can be covered by only a few classifiers. However, most plots also have a great number of outliers. Thus, XCSF needs additional classifiers to also cover these niches.

Performance measure results

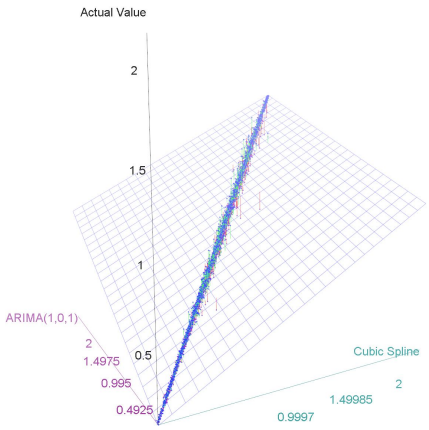
Table 5.3 summarises the experimental results for SMAPE, MASE, and Theil's U-statistic for each time series, forecast method, and combination strategy. The average ranking for each strategy is also given in the last column, labelled "Avg. rank". For each forecast accuracy measure, the strategies are ranked accordingly. As you can see, ARIMA(1,0,1) provides rather accurate forecasts, whereas the forecasts of cubic spline typically result in higher forecast errors. A good combination strategy should therefore tend to assign higher weights to ARIMA(1,0,1), not to cubic spline. Although SA does not address this point by assigning equal weights to both forecast models, it ranks third best after XCSF and ARIMA(1,0,1). Concerning the U-statistic and the MASE measure, a forecast is considered precise in case its value is lower than 1. The results show that XCSF is often better than this reference value, whereas all other combination strategies are mostly above this threshold. XCSF outperforms the two individual forecast techniques for every time series, improving the forecast accuracy by assigning higher weights to ARIMA (see table 5.4). Independent of the characteristics of the time series (e.g. stationary or non-stationary), XCSF mostly performs similar or slightly superior to the best reference technique.



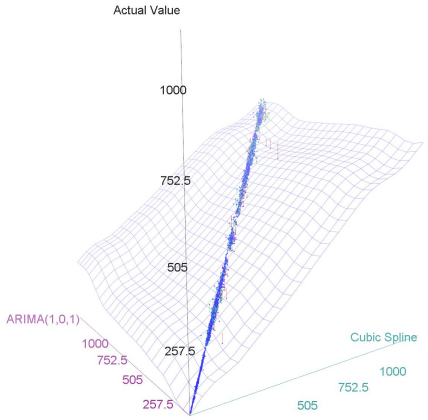
(a) BARB-MSCI



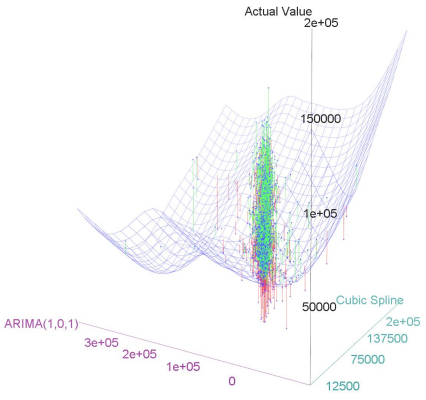
(b) BBK



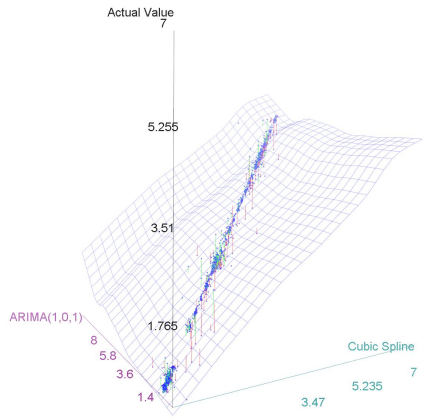
(c) FRED-CRES



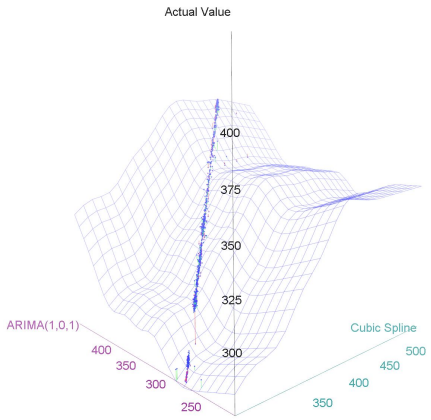
(d) FRED-GOLD



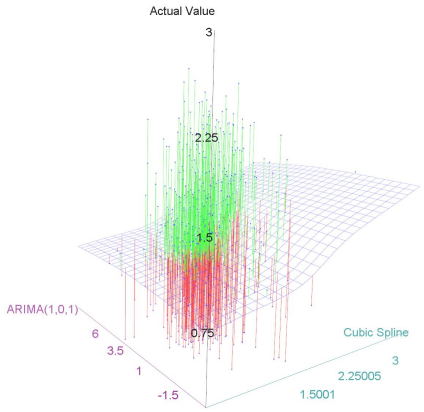
(e) FRED-MKT



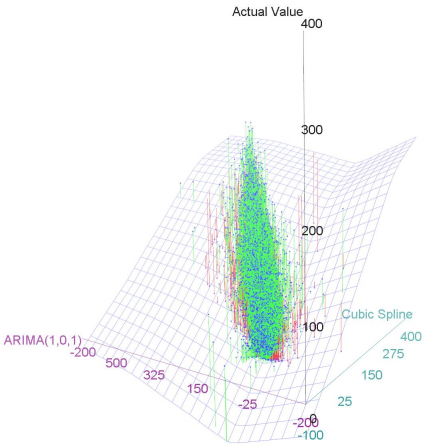
(f) FRED-RIFS



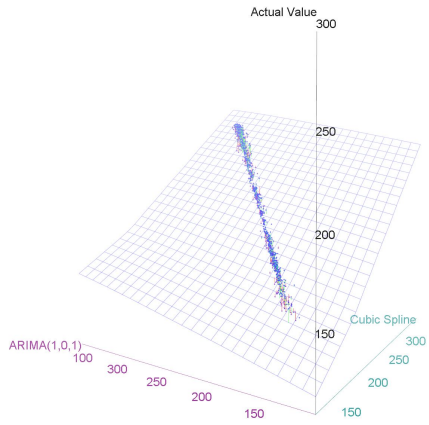
(g) LIVEX-LVX50



(h) PSYCH-XIV



(j) SUNSPOTS



(j) SPDJ-SPFTR

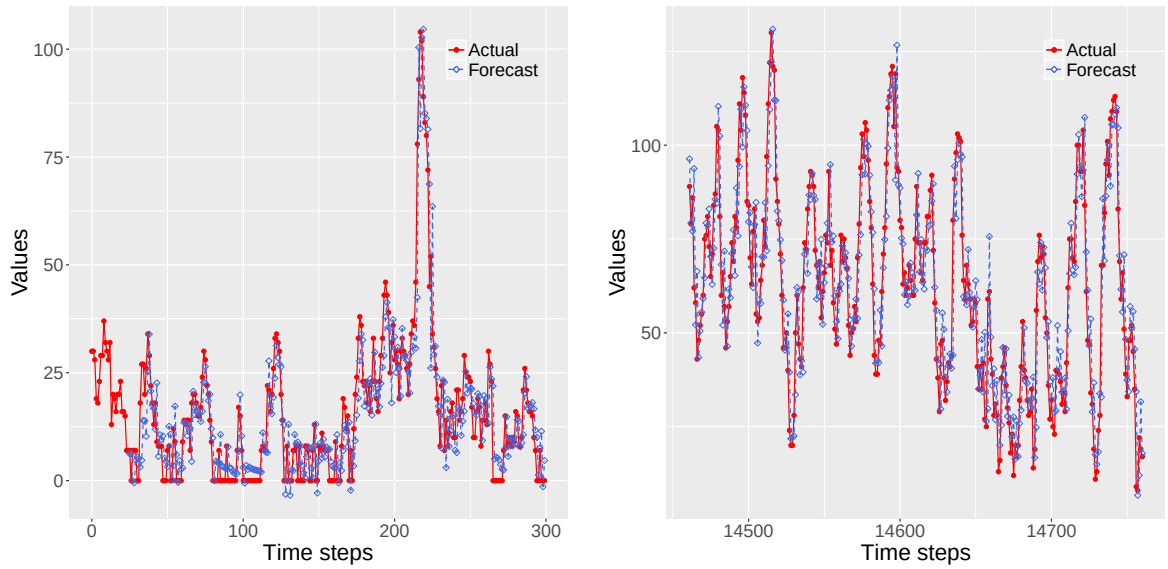
Figure 5.3: Scatter plots with a fitted regression surface showing the problem space for the combination of two forecast methods derived by ARIMA(1,0,1) and cubic spline.

Table 5.3: Results per time series for an ensemble with two forecast methods: ARIMA(1,0,1) and cubic spline(CS) (bold values highlight the best performances). Results are sorted by average rank.

	MSCI	BBK01	CRES	GOLD	MKT14	RIFS	LVX50	PSYCH	SUNSP.	SPFTR	Avg. rank
U-statistic											
XCSF	0.99	**0.90**	**0.98**	1.27	**0.48**	**1.01**	**0.16**	**0.86**	**1.09**	**0.37**	1.5
ARIMA	1.18	1.12	1.04	**1.14**	0.84	1.04	1.38	1.30	1.12	1.12	2.7
SA	1.02	1.11	1.03	1.18	1.44	1.15	1.06	1.15	1.15	1.12	3.0
OP	**0.97**	2.21	1.20	1.18	1.22	1.10	1.91	1.00	1.18	1.12	3.5
OW	1.18	1.51	1.08	1.18	1.37	1.07	1.05	1.63	1.18	1.15	3.8
CS	1.22	1.34	1.32	1.48	2.07	1.64	1.54	1.97	1.45	1.38	5.8
SMAPE											
ARIMA	**73.37**	0.39	1.58	**0.47**	5.19	**3.61**	0.11	37.93	21.15	**0.40**	2.2
XCSF	88.66	**0.37**	1.53	0.61	**4.90**	**3.61**	0.11	**34.39**	**20.06**	**0.40**	2.5
OW	74.66	0.42	**1.39**	0.48	6.35	3.76	**0.10**	44.98	20.95	0.41	3.0
SA	74.13	0.38	1.50	0.49	5.91	3.89	**0.10**	42.31	21.32	0.41	3.2
OP	74.58	0.47	1.59	0.48	5.73	3.88	0.12	41.58	21.63	**0.40**	3.8
CS	75.19	0.46	1.49	0.60	7.80	5.11	0.13	48.68	22.32	0.50	5.3
MASE											
XCSF	**0.69**	**1.04**	**1.02**	**1.12**	**1.02**	1.16	**1.00**	**0.79**	**1.08**	**0.98**	1.1
ARIMA	0.78	1.09	1.11	1.16	1.06	**1.15**	1.14	0.86	1.13	1.13	2.4
SA	0.92	1.06	1.03	1.18	1.20	1.19	1.24	0.99	1.12	1.15	3.4
OP	0.86	1.46	1.07	1.17	1.16	1.32	1.43	0.94	1.14	1.14	3.9
OW	1.05	1.24	1.04	1.17	1.30	1.20	1.20	1.11	1.14	1.15	4.0
CS	1.28	1.28	1.14	1.47	1.64	1.52	1.59	1.35	1.34	1.41	5.9

Accuracy of the predictive model

The time plots in figure 5.4 exemplarily depict the development of the accuracy of XCSF over time, for the sun spots data set. The initial 24 values have no associated forecasts, as ARIMA needs a warm-up phase of 25 values to build an initial model of the time series. Afterwards, both forecast methods provide one-step forecasts for every data point of the time series. In the beginning, we can clearly see deviations between the actual observations and their forecasts. However, if we look at the plots showing the last 300 observations, the forecasts almost perfectly fit the time series. The behaviour of the time series is the same in both situations (several peaks and trend changes), however the forecast accuracy is much better in the end. As the forecast methods itself are not able to learn, this observation shows that XCSF is able to improve the forecast accuracy over time.

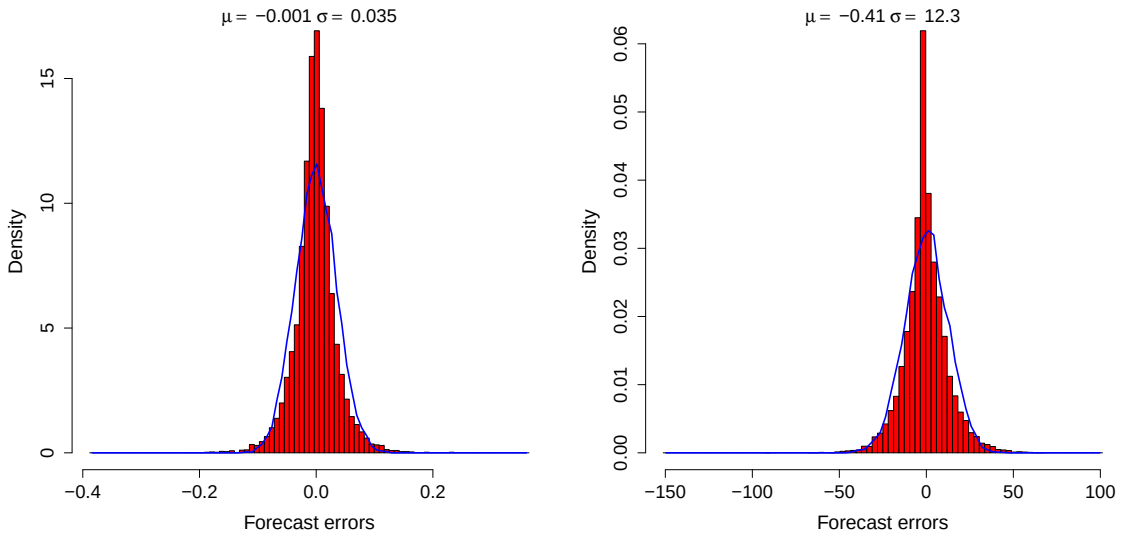


(a) First 300 values **(b)** Last 300 values

Figure 5.4: Time plots showing the actual observations (red solid line) and the forecasts (blue dashed line) for the sun spots data set.

To confirm the reliability of our model, we have to look at the error correlations for successive forecasts, also confirming that the errors are normally distributed with mean zero. The representative histograms of the forecast errors in figure 5.5 support the observation that the distribution of forecast errors is centred around zero, and approximately normally distributed. The mean forecast errors are slightly below zero.

Additionally, a Ljung-Box test is carried out to determine whether there is significant evidence for non-zero correlations at lags 1 to 20. The Ljung-Box test returns p-values of $p \approx 2.2e^{-16}$ (BBK and sun spots), indicating that there is little evidence for non-zero autocorrelations in the forecast errors for lags 1 to 20. These observations hold for the other time series as well. Therefore, a tentative conclusion can be drawn that XCSF provides an adequate predictive model.



(a) BBK **(b)** Sun spots

Figure 5.5: Forecast error histogram for XCSF (BKK and sun spot data set).

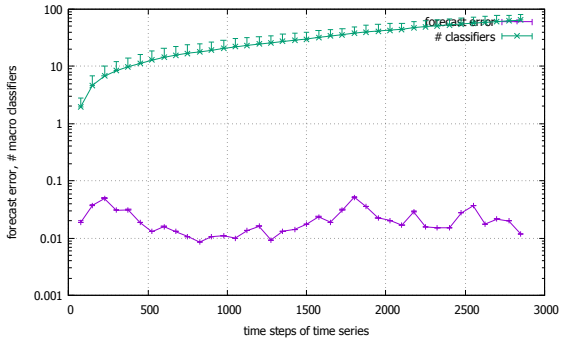
XCSF's learning behaviour

Figure 5.6 depicts the development of two XCSF-specific metrics. Each plot depicts the averaged system error that corresponds to the mean absolute (forecast) error, as well as the number of classifiers of the population $[P]$. Each data point is the average of 75 consecutive execution steps. The curves show the development of the corresponding means and standard deviations over 30 i.i.d. experimental runs with different random seeds to face the bias that occurs due to randomized operators (e.g. crossover, mutation, covering) XCSF relies on.

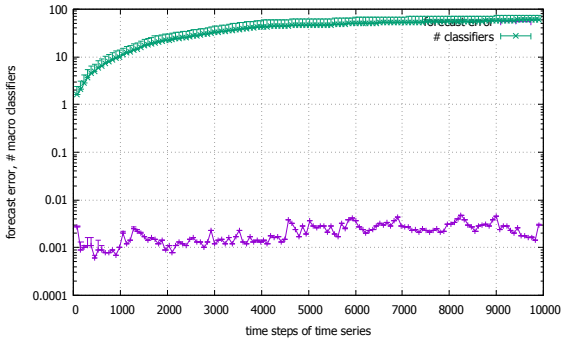
The time plots show that XCSF is able to approximate the underlying combination problem accurately. Most of the time, the error rate lies beyond 1% for all ten time series. The forecast error continuously fluctuates between 0.01 and 0.001. Therefore, XCSF approximates accurate classifiers, whereas a classifier is defined as accurate if the error lies below a threshold defined by $\epsilon_0 = 0.01$.
The number of physically stored classifiers (macro-classifiers with numerosity $cl.num > 1$) converges to about 100. As can be seen, the chosen population size restriction of $N = 200$ micro-classifiers is sufficient to approximate the problem function illustrated in figure 5.3. Higher values (up to 6400) did not lead to smaller forecast errors. The time series SPDJ-SPFTR (Figure 5.6j) and LIVE-LVX (Figure 5.6g) seem to resemble easier problems, since they only need about 15 to 20 classifiers. The sunspot time series is the only series where XCSF evolves more than 100 macro classifiers (Figure 5.6i). The number of peaks and valleys correlates with the seasons in the SIDC-SUNSPOTS time series (cf. figure 5.2). We attribute the fluctuation effect to the particular learning task XCSF is confronted with. The goal XCSF is striving for is not to forecast the next time series value directly, but to learn, to which of the individual forecast methods to assign a higher weight in which situation. Thus, when both single methods are forecasting values with a high error, XCSF also outputs a combined value that leads to an error with

a similar (nonetheless often smaller) amount of forecast errors. However, it its visible that the absolute error is reduced over time.

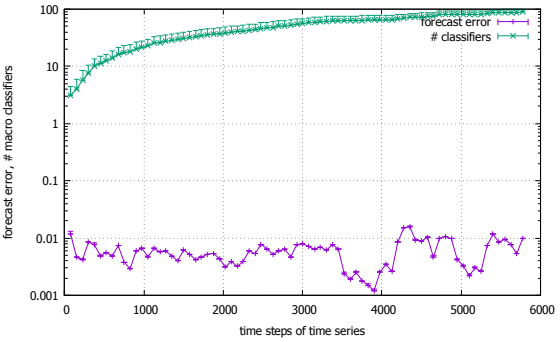
Time series that are characterised by trends seem to be a challenge for XCSF (see FRED-GOLD in figure 5.6d). In this case, XCSF rarely gets a chance to reinforce already evolved classifiers, since they are seldom triggered again due to the trend. Put differently, when a time series follows a trend, the already covered niches within the approximated problem function will not or rarely be sampled again. Thus, XCSF is not able to learn an adequate weighting for these situations. Nonetheless, XCSF is deemed as a promising candidate for the ensemble time series forecasting problem.



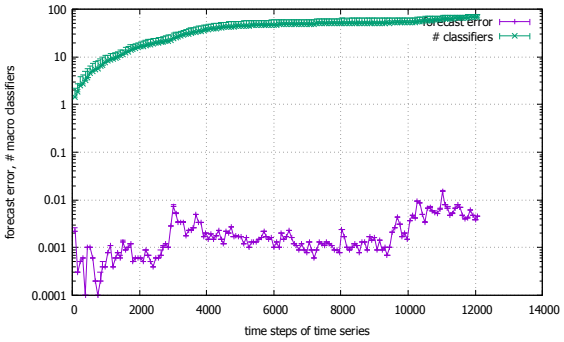
(a) BARB-MSCI



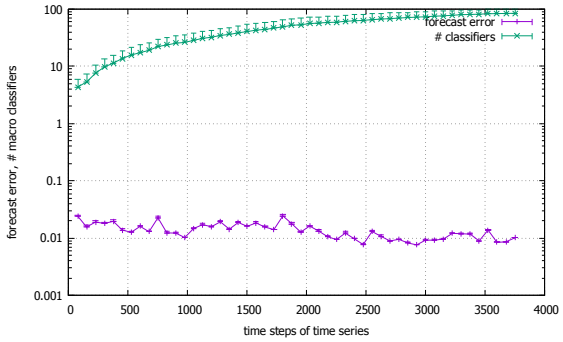
(b) BBK



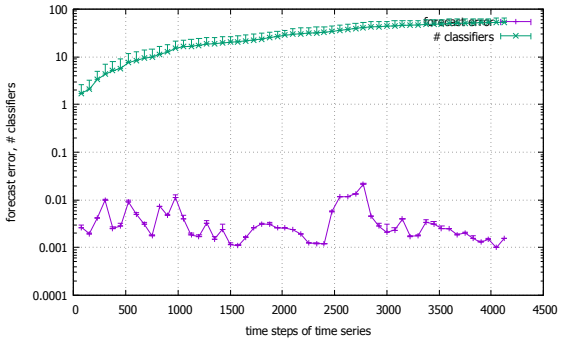
(c) FRED-CRES



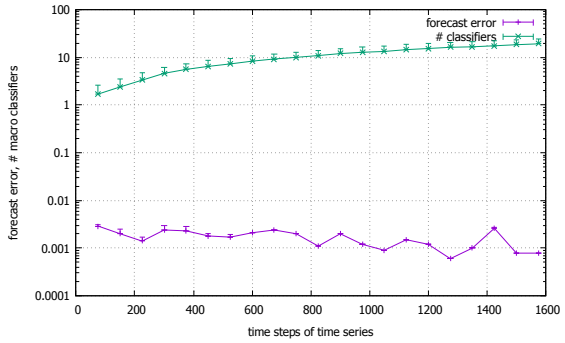
(d) FRED-GOLD



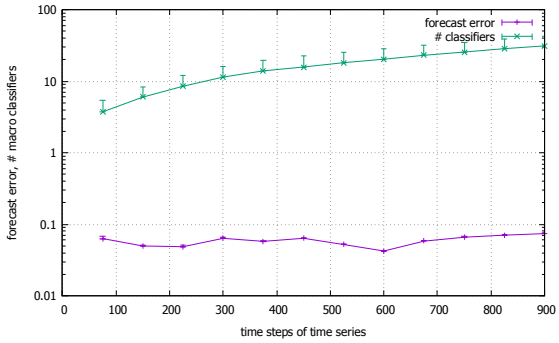
(e) FRED-MKT



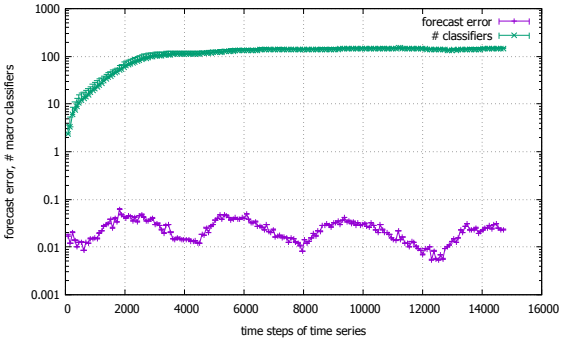
(f) FRED-RIFS



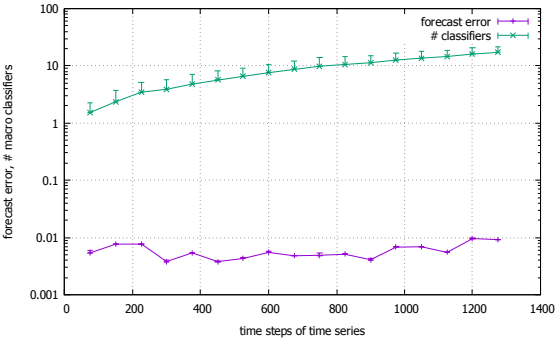
(g) LIVEX-LVX50



(h) PSYCH-XIV



(i) SUNSPOTS



(j) SPDJ-SPFTR

Figure 5.6: Time plots illustrate the learning process of XCSF for a combination of two forecast methods. The plots show the development of XCSF's forecast error and its population size in number of macro classifiers over time.

Depiction of XCSF's evolved classifiers

Table 5.4 lists the most influential classifiers (with the highest fitness) out of the final population of 83 classifiers, evolved after 9000 time steps for the BBK time series. For each classifier, its condition, its coefficient vector $\vec{w}$, its fitness ϕ, its experience, and its prediction error ϵ[3] are presented.

Looking at the individual conditions, it can be clearly seen that each four of the five listed classifiers cover approximately the same region of the problem space. Put another way, the centre and stretch values of the hyper-ellipsoids are rather similar. We attribute this effect to the smaller amount of opportunities of the GA to refine the condition structures, since the learning task only comprises $\approx$ 9000 steps. Another reason originates from the shape of the underlying problem function as depicted in figure 5.3. Indeed, it seems to be possible to only create one single hyper-ellipsoid with a linear approximation to resemble the regression surface. However, to find a nearly optimal condition structure, XCSF needs more than one classifier to allow for exploratory behaviour exerted by the GA. The classifier that fits this particular problem function best is expected to outperform any competing classifiers in terms of its fitness, its numerosity, and its experience. These classifiers assign an average weight of $w_1 \approx 0.7$ to ARIMA, and an average

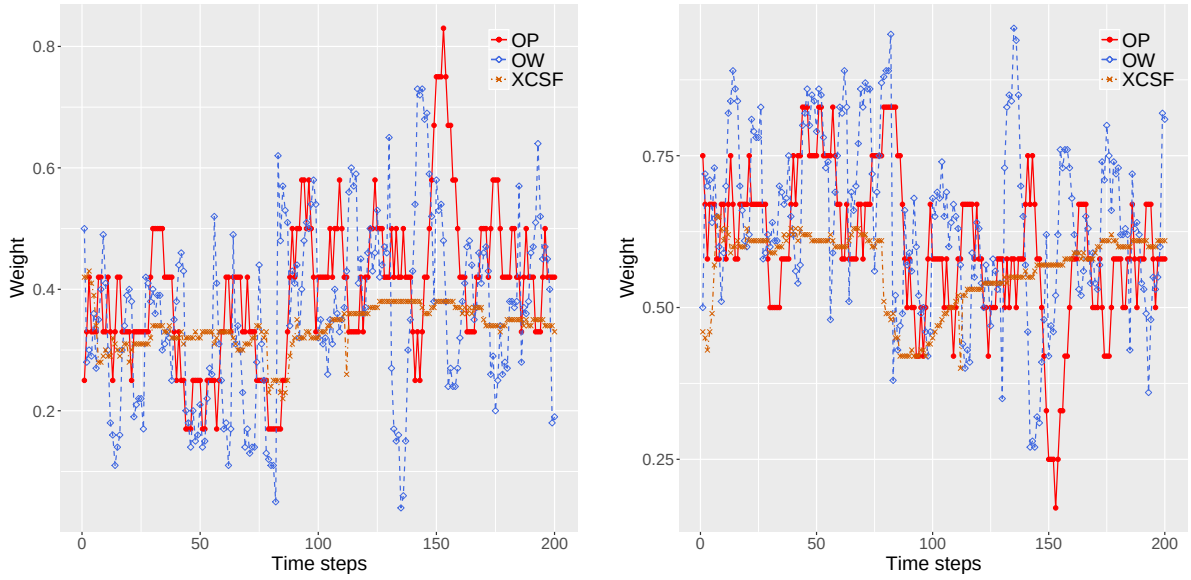
[3] ϵ values are to be multiplied by 10^{-3}

weight $w_2 \approx 0.3$ to cubic spline. This confirms that XCSF is able to learn to assign higher weights to forecast methods with higher accuracy. The fifth classifier is rather specialised, only covering a small region. It assigns weights of 0.594 and 0.406 to ARIMA, respectively cubic spline. In comparison, optimal weights gives an average weight of 0.67 ($\sigma = 0.16$), outperformance assigns an average weight of 0.58 ($\sigma = 0.17$) to ARIMA.

Table 5.4: XCSF population showing the most important classifiers with the highest fitness for the BBK time series and two forecast methods (sorted by fitness ϕ ascending).

Cond.: Center / Stretch / Angle	**Coefficients** $\vec{w}$	ϕ	**Num.**	**Exp.**	ϵ
0.566, 0.566 / 1.243, 0.360 / 3.68	0.733, 0.266	0.437	24	8685	0.0022
0.566, 0.566 / 0.678, 0.360 / 3.68	0.726, 0.274	0.261	29	8645	0.0020
0.566, 0.566 / 0.678, 0.359 / 3.55	0.724, 0.275	0.192	22	8085	0.0032
0.566, 0.566 / 0.356, 0.360 / 3.68	0.732, 0.268	0.151	19	7059	0.0030
0.659, 0.660 / 0.173, 0.072 / 0.39	0.594, 0.406	0.111	8	2474	0.0029

Figure 5.7 depicts the development of the weights assigned to cubic spline and ARIMA by OP, OW, and XCSF. For OW and OP, the constraint holds that the sum of the weights assigned to both methods sums up to one for each time step. This is not necessarily true for XCSF, which can assign weights more freely. Compared to cubic spline, the average weight assigned to ARIMA is higher. However, the actual weights are changing faster for OW and OP, whereas XCSF slowly adapts the weights. In case of a significant change in the behaviour of the time series where the previously better method becomes the worse method, OW and OP can adapt faster, since XCSF needs some time to adapt its rule base. However, it is assumed that when a similar change occurs a second time, XCSF should have generated classifiers for this special situation.



(a) Cubic spline **(b)** ARIMA

Figure 5.7: The weights assigned to cubic spline and ARIMA by OW, OP, and XCSF are compared for the first 500 time steps of the Sunspot data set.

5.3.3 Experimental results: More than two forecast methods

In the following, the combination strategies are evaluated for higher dimensions, e.g. for ensembles of size 3, 5, and 7. The ensembles comprise the following forecast methods: 3 = ARIMA + CS + MA; 5 = ARIMA + CS + MA + RW + ETS, and 7 = ARIMA + CS + MA + RW + ETS + DES + ES. These combinations were chosen for their structural diversity between the different forecast methods.

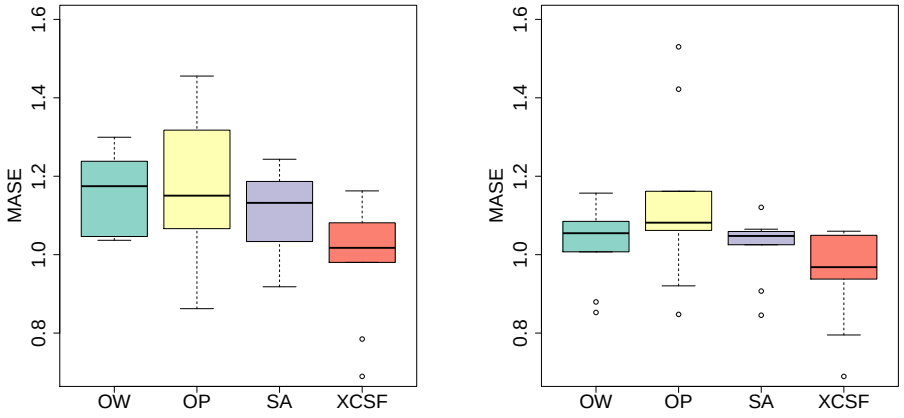
Table 5.5 summarises the experimental results. The results are averaged over all ten time series. Consequently, set size 2 resembles the average results of table 5.3. Our findings show that a combination consisting of a higher number of forecast methods improves the forecast accuracy. The lowest MASE, U-statistic, and SMAPE values were achieved using five and seven forecast methods. However, a combination of more than five methods did not significantly improve the performance, which is consistent with findings of [Arm01]. Considering MASE and the U-statistic, XCSF has the lowest values. In contrast, for the SMAPE measure, it ranks last. However, the difference to the other strategies is rather small. This effect can be ascribed to the explorative phases of XCSF, where higher deviations from the actual time series value may occur.

Table 5.5: Average SMAPE, U-statistic, and MASE for several sets of forecast methods with different size (bold values highlight the best performances).

Strategy	**Set size**				**Avg. rank**
	2	3	5	7	
U-statistic					
XCSF	**0.81**	**0.76**	**0.73**	**0.73**	1.00
SA	1.14	1.02	0.99	0.98	2.00
OW	1.24	1.05	1.02	1.01	3.00
OP	1.31	1.18	1.08	1.05	4.00
SMAPE					
OW	15.35	**14.13**	**14.34**	**14.19**	1.50
SA	**15.04**	14.59	14.42	14.41	1.75
OP	15.05	14.67	14.57	14.48	3.25
XCSF	15.46	15.21	15.16	15.19	4.00
MASE					
XCSF	**0.99**	**0.95**	**0.93**	**0.93**	1.00
SA	1.11	1.02	0.98	0.97	2.00
OW	1.16	1.03	1.01	0.99	3.00
OP	1.17	1.13	1.06	1.03	4.00

Figure 5.8 shows the results of the MASE forecast accuracy measure for several set sizes, averaged over all ten time series. The box plots show the statistical distribution of MASE over all time series. The bottom and top of each box represent the first and third quartiles, and the band inside the box represents the median. Outliers are indicated by separate points. Independent of the number of forecast methods combined, the plots show that XCSF has the lowest average MASE, while also having the lowest lower quartile. For larger set sizes, MASE is reduced

independent from the combination strategy. However, the difference between five and seven forecast methods is rather low. This finding is in accordance with findings of [Arm01]. Thus, we can safely assume that adding more forecast methods will not improve the forecast accuracy any more.



(a) 2 methods **(b)** 3 methods

(c) 5 methods **(d)** 7 methods

Figure 5.8: Box plots showing the MASE error for set sizes of 2, 3, 5, and 7 forecast methods, averaged over all ten time series (lower values are better).

Additional time plots illustrating the learning process of XCSF for a combination of more than two forecast methods can be found in the appendix in figures A.1, A.2, and A.3.

5.4 Summary

This chapter has presented how the extended classifier system for function approximation (XCSF), a genetic, rule-based algorithm, can be applied to the challenging task of multi-model ensemble time series forecasting. First, the general problem was formalised. Second, we applied XCSF to the problem and its modifications were explained. Finally, the results of my tests showed that XCSF is superior to other well-known combination strategies. The evaluation

study was done with real-world time series exhibiting different characteristics. An in-depth investigation of the proposed concept with regard to XCSF-specific metrics and issues, such as the classifier evolution, and parameter studies for the most important learning parameters were executed. Using this setup, it was demonstrated that the utilisation of XCSF as a forecast combination approach is very promising in comparison to the reference methods. Experiments with more than two forecast methods show that the multi-model combination with XCSF is also favourable for higher problem dimensions.

The following chapters introduce two case studies for the utilisation of forecasts within OTC. First, the adaptation process of signalisation is enhanced by incorporating forecasts of future traffic flow. Second, two route guidance algorithms for urban road networks which compute the proposed routes based on the estimated future travel times are presented.

Chapter 6

Forecast-augmented anticipatory adaptation of green times

The performance of traffic control in urban areas is highly dependent on a near-to-optimal setting of the signal plans. However, the design of fixed-time signal plans is a complex and time-consuming task. It is usually executed by traffic experts, which define green times, clearance times, the number of phases, and much more. Competing dominant traffic flow, densely spaced intersections, and additional influences by nearby signalised intersections make it a complex optimisation problem. Autonomously adapting signalling strategies to changing traffic demands in urban areas has been used as one application scenario for real-time traffic management systems. Striving for the ability to cope with the dynamic behaviour of traffic and to react appropriately to unforeseen conditions, such solutions dynamically adapt the signalisation to the monitored traffic demands.

Traffic management systems are usually divided into two categories: fixed-loop and adaptive systems. The first category relies on a fixed-time signalisation with a limited set of predefined signal plans for each signalised intersection. These signal plans are developed by traffic experts based on historic traffic profiles. In consequence, the system is not responsive to the dynamic and changing traffic demands. Furthermore, due to steady increase in traffic demand and changes in traveller's patterns, the signal plans become outdated over time. Adaptive systems alter the signal timing based on the observed state of traffic. The traffic demands are monitored in real-time by sensors along the road and serve as input for the signal time optimisation at each intersection. Some traffic management systems react to the traffic measured in the previous interval [PRT+08] while others adjust the signalisation proactively to the estimated traffic demand in the near future [SU10]. This continuous adaptation leads to a better system performance, even in case of changing traffic demands. Often, the adjustment is executed repeatedly in fixed intervals (e.g. every five minutes), respectively after one to three full cycles of the active signal plan [SU10]. Making forecasts of future developments and altering the control strategies in accordance to these predictions can lead to a better system performance. To the best of the author's knowledge, there is no deployed ATCS that actually incorporates traffic forecasts in its optimisation process.

In the following, OTC-Pro is introduced, a proactive extension of the self-adaptive traffic control system OTC, which was explained in section 2.2. OTC-Pro uses the time series forecast module from section 2.4 to make short-term forecasts of the future traffic flow. The forecasting process follows the idea of multi-model ensemble forecasting, combining individual forecasts of several forecast methods. These forecasts are used to proactively adapt the signal plans at signalised

intersections to current and future demands. This is a challenging task, as traffic flow in urban areas is not normally distributed and can exhibit chaotic behaviour. Consequently, traffic flow on signalised urban arterial roads cannot be predicted as accurately as on motorways [SK03]. In section 6.3, a highly self-adaptive technique is presented to relieve the traffic engineer from complex design-time situated tasks related to signal plan design, forecast method selection and configuration. Based on this technique, an approach is implemented that makes reliable forecasts despite the non-linear and non-stationary behaviour of traffic flow. The chapter closes with a simulation study in section 6.4 and a short summary of the findings.

6.1 Related Work

Anticipatory traffic light control touches a number of topics of different research disciplines including traffic-adaptive control systems and time series forecasting. For terms and definitions concerning fixed-time signalisation and traffic engineering the reader is referred to section 2.1. Section 3 offers a general introduction to time series forecasting.

6.1.1 Traffic-adaptive control systems

In the last decades, several traffic-adaptive systems have emerged. *Traffic-adaptive control systems*, such as OPAC [Gar89], SCATS [SD80], and SCOOT [RB91] (the two most widely used ATCSs) adapt green times, cycle times, and/or phase sequences according to the recently monitored traffic demands. The adjustment is executed repeatedly every five to ten minutes, respectively after one to three full cycles of the signal plan [SU10]. The current traffic demands are usually represented by the number of vehicles passing a sensor at the respective signalised intersection, the current queue length in front of a traffic light, or the occupancy level of a detector. The non-dimensional occupancy rate indicates the ratio over a certain time interval $\triangle t$ where a loop detector is occupied by passing vehicles. It is calculated by:

$$occ(x,t) = \frac{1}{\triangle t} \sum_{\alpha=\alpha_0}^{\alpha_0+\triangle N-1} (t_\alpha^1 - t_\alpha^0) \tag{6.1}$$

where α denotes a vehicle, t_α^0 is the time where the front of the vehicle passes the detector, t_α^1 is the time where the tail of the vehicle passes the detector, and $\triangle N$ is the traffic flow as the number of vehicles.

SCATS: The *Sydney Coordinated Adaptive Traffic System* (SCATS) [SD80] uses up-to-date flow and occupancy data collected at each intersection to process a region-wide optimisation of cycle times, phase splits, and offsets. It groups adjacent intersections into so-called subsystems which are managed together. The network is structured hierarchical. A central management computer manages global data and access control, regional controllers determine phase sequences and their durations, whereas local controllers can terminate and skip phases. An estimated number of about 42000 intersections in over 154 cities in 25 countries are controlled by SCATS.

OPAC: The *Optimized Policies for Adaptive Control* (OPAC) system [Gar89] resembles a real-time, traffic adaptive control of traffic signals in a distributed fashion by individual controllers. The OPAC controllers are able to adapt the cycle time of signal plans within predefined boundaries. Flow profiles for each approaching lane of an intersection serve as input for the cycle time adaptation. The head of such a flow profile consists of actual volume counts from upstream detectors, whereas the profile's tail is a smoothed projection into the near future. These profiles estimate the development of queues in front of traffic lights and can trigger phase extension up to two seconds.

SCOOT: SCOOT [RB91] is designed in a centralised fashion to employ system-wide strategies for changes in signal timings. Every five minutes, the phase splits, offsets, and cycle times are adjusted according to the current volume and occupancy states. Additional components automatically detect traffic incidents and analyse and display traffic information. The system works best for networks having a grid layout.

These *responsive adaptive* systems [SS10] adjust signal timings based on data collected over a certain time span (several minutes or cycles) before implementing a timing chance. This responsive process has the disadvantage that the adaptations lag behind current traffic demand. In contrast, by following a proactive approach, this drawback is omitted by making short-term forecasts of the traffic flow and adapting green times and cycle times with respect to the current and future demands.

6.1.2 Short-term traffic forecasting

Several surveys deal with the topic of short-term forecasting of traffic flow, pointing out that making forecasts of future traffic developments is an important aspect for *intelligent transportation systems* (ITS) [VGK03, BF12, BAR15]. Typically, forecast techniques takes the last recorded traffic flow into account to forecast the estimated traffic conditions of the near future. Some techniques rely on aggregated historical information (e.g. daily load curves [CKWS04]), or a combination of both historical and real-time data. Furthermore, there are advanced forecast techniques, such as kalman filters [GLW97] or artificial neural networks [DC97]. They allow to consider additional input parameters, such as traffic density, weather conditions, or data from adjacent roads. Novel approaches also apply machine learning concepts to this problem [ERC16]. A collective consideration of several forecast models is a more powerful approach than just relying on one individual technique [AA14]. Consequently, some researchers propose to follow the idea of ensemble forecasting, combining the forecasts of several independent techniques [YLS+12] to forecast the traffic flow on highways. Actually, most works focus on traffic flow forecasting on highways [CC07] which is considered to be less complex than forecasting of traffic flow within urban areas. There are only few researchers that propose forecast techniques for urban transport networks [GBO09, WZW+16].

6.2 Problem statement

Adapting signal timings at a signalised intersection according to future demands creates two problems that have to be solved. First, reliable short-term forecasts for each turning have to be created, representing the traffic developments in the next few minutes. Related to this problem is the choice of input for the individual forecast methods. Second, these forecasts have to be incorporated into the self-adaptive adaptation strategies of the OTC controller, according to their reliability. Here, reliability is defined by the result of a metric measuring the accuracy of the forecasts.

The first task is especially challenging, as traffic flow in urban traffic networks is non-linear and non-stationary. In contrast to traffic flow profiles on highways, urban flow profiles are much more complex due to signalisation and influences from incoming and outgoing sections. Due to their chaotic behaviour, the resulting time series are difficult to model and to forecast. Considering the design of the input vector, the input situation sit can be modelled as a vector of values, derived from one or more sensors, monitoring the underlying system (in this case the road traffic network). This approach can be extended, not only taking one value, but to use a list of the last x historic sensor values. Different methods to specify the situation are possible:

1. In terms of road traffic, the most simple approach to define the situation is to use the observed traffic flow as input. Hence, the observation of a traffic flow of $m\frac{vehicles}{hour}$ for the considered turning can be used. The advantage of this solution is the non-complex learning strategy and a feasible list of stored experiences. The disadvantage is that neither the history that resulted in this traffic conditions nor the dynamics of the traffic flow is considered.

2. Intuitively, using the last x consecutive traffic flow measurements provides a better basis than just using the currently monitored flow. This corresponds to the basic process model of a variety of forecast techniques: Using the last x observed traffic flow values as input, the best configuration of weights is predicted instead of the next traffic flow. Compared to the first solution, a significantly larger number of situations has to be considered due to combinatorics – resulting in a condition space that grows as a function of the number of historical flow that is taken into account. The disadvantage of the previous solution – i.e. neglecting the historical context and the dynamics of traffic – are suppressed, but still no classification of the general learning problem is achieved, which is the ultimate goal of machine learning [Mit96].

3. In order to derive more generalised classes of situations, the absolute values of the observations have to be replaced by an abstracted categorisation. Instead of working on absolute flow values, more generalised indicators are used to define the conditions. For instance, the percental change for the measuring points, an extrapolated trend, or statistical indicators of the forecast errors (i.e. standard deviation of forecasts, variation, etc.) can serve as basis for the situation description.

Considering the second task, OTC's observer on layer 1 has to be extended by the previously introduced forecast module for time series. The controller selects and executes signal plans based on a situation description of the traffic demands given by its associated observer. Until now, this input was defined by a vector $\vec{M}_t$ containing the traffic flow for the current time step t,

measured in vehicles per hour, for each of the intersection's turning movements. By making use of our time series forecast module to forecast the upcoming traffic flow of each turning, future traffic conditions can be considered. In the following, $\vec{F}_{t+m}$ represents these forecasts. Having created two vectors, $\vec{M}_t$ and $\vec{F}_{t+m}$, two options of how to pass the situation description to the controller have be considered.

Extending the controller interface: The first option is to give both vectors to the controller as input. This would result in an adaptation of the controller's interface, and in an alternative representation of its rule base. In terms of the adapted extended classifier system used within OTC, the condition of the rules would have to be adapted. But more importantly, a more complex situation description increases the dimensionality of the problem space that the learning component of the controller would have to explore. Consequently, the controller would need more rules to represent the problem space sufficiently. This can lead to a degraded system performance and a longer learning process.

Combining both input vectors: Following this train of thought, a possible solution should not complicate the controller's interface. The second option involves a combination of both, the current and the future, situation vectors. One possibility is to combine $\vec{M}_t$ with the estimated future developments $\vec{F}_{t+m}$ as

$$\vec{C} = \alpha * \vec{M}_t + (1 - \alpha) * \vec{F}_{t+m} \tag{6.2}$$

meaning that for each entry i of both vectors, the combined result $\vec{C}_i$ is calculated as $\alpha * \vec{M}_i + (1 - \alpha) * \vec{F}_i$. Afterwards, the resulting vector $\vec{C}$ is given to the controller as usual. Thus, the controller does not need any adaptations at all. A crucial point is the selection of the optimal weight factor α. This factor can be static, or it can be dynamically calculated based on the accuracy of the latest forecasts. Consequently, a performance measure is needed that defines the forecast accuracy of each individual turning's forecast module. The higher its accuracy, the more likely it is that future forecasts will be equally accurate. Therefore, higher accuracy leads to more influence of the forecasts on the combined situation vector.

In the following, the second approach is pursued. A detailed explanation on the practical implementation is given.

6.3 Adapting green times with forecasts of turning movements

OTC runs in cycles – meaning that a review of the current control strategy takes place at the beginning of each second or third cycle of the currently active phase plan (for traffic safety reasons and to allow the current strategy to show its performance). Taking into account a standard duration of such a phase plan of about 90 seconds, OTC decides about an adaptation in intervals of three to five minutes. Hence, an appropriate forecast would alter the adaptation strategy of OTC: Instead of selecting the most promising signal plan for the current traffic conditions, selecting the one for the estimated traffic demands in the near future (5 to 15 minutes) can lead to a better anticipatory adaptation strategy. Historical and currently monitored traffic flow of

turning movements is used to estimate the most probable traffic state at a certain future point in time. This process involves the following four steps:

1. Create a vector describing the current traffic demands.
2. Create short-term forecasts for future traffic demands.
3. Combine these vectors into one forecast-augmented situation description.
4. Use the new vector to select the next signal plan.

Analogously to the previously described steps, the sequence diagram in figure 6.1 depicts the sequence of interactions of an OTC-Pro controller to derive a forecast-augmented situation of the local traffic conditions. The diagram shows the entities involved in the process, and the sequence of messages exchanged between these objects. Vertical lines represent the life lines of the entities, horizontal arrows represent the messages exchanged between them.

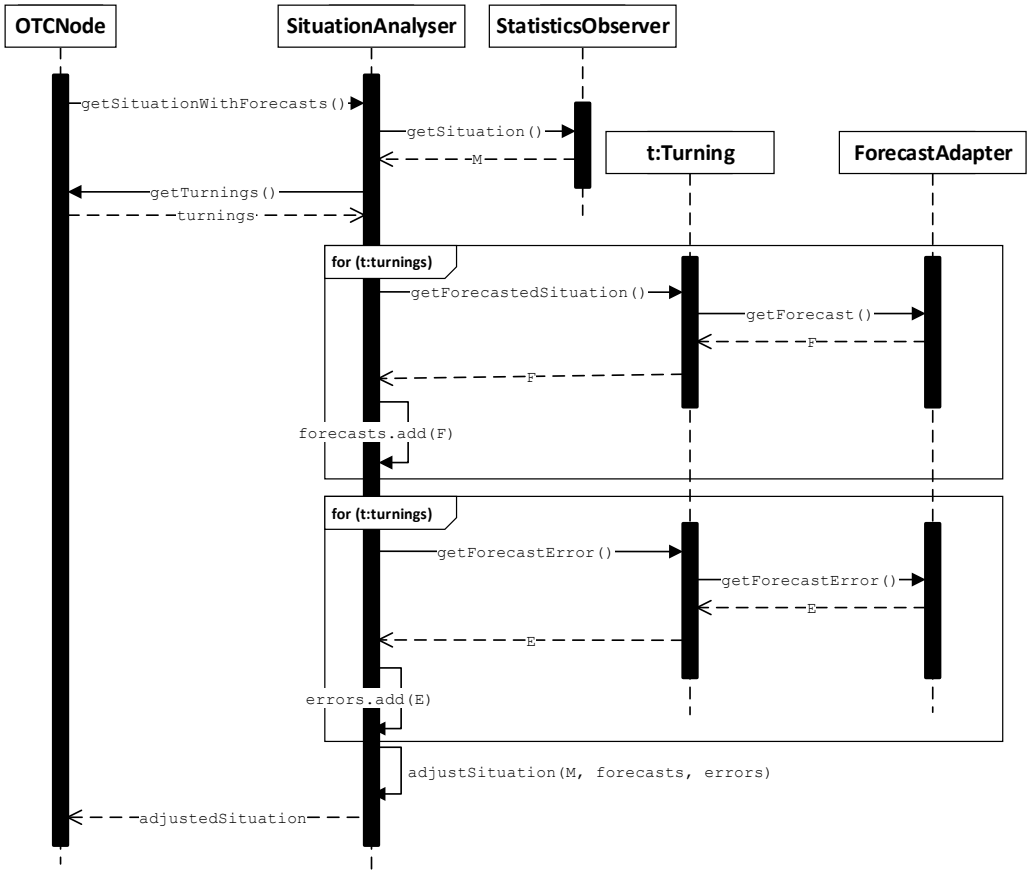


Figure 6.1: A sequence diagram showing the process of deriving a forecast-augmented traffic situation description.

First, an *OTCNode* (representing the internal model of a physical intersection) demands a situation vector $\vec{M}$, containing the traffic demands for all its associated turnings. The request is forwarded to the *SituationAnalyzer*. The *StatisticsObserver* derives the latest actual measurements from detectors located at the intersection and returns a vector comprising traffic flow

values (in vehicles per hour) for each turning. Afterwards, for each turning t, a forecast is derived from the associated *ForecastAdapter*. The forecasts are created for the point in time when a new signal plan can be activated. Each forecast F is then added to the vector $forecasts$. A similar process is executed to derive the average forecast error for each *ForecastAdapter*. The forecast performance of each *ForecastAdapter* is measured based on its latest forecast errors. Finally, the method *adjustSituation* is called with vectors containing the current situation $\vec{M}$, the forecasts $forecasts$, and the forecast errors $errors$. The adjusted situation is returned to the *OTCNode*. Finally, the new situation vector is used to select a signal plan to be activated in the next cycle.

6.3.1 Representation of the current traffic demands

The intersection's momentary traffic flow is estimated based on a real-valued vector $\vec{M} = (M_1, \ldots, M_n)$ containing the estimated hourly flow M_i for each of the intersection's n turning movements. Figure 6.2 depicts a class diagram with the entities involved in the creation of $\vec{M}$, and their relations among each other. An *OTCNode* resembles OTC's abstracted internal model of the physical intersection. Therefore, it knows all its related turnings and detectors, the approaching and outgoing sections, and the current signal parameters. A description of the monitored traffic flow M_i for each turning i of the respective intersection can be derived via the according *SituationAnalyzer*. The history of each turning's traffic demands is available as time series via the *StatisticsObserver* module which retrieves data from physical sensors in fixed time intervals.

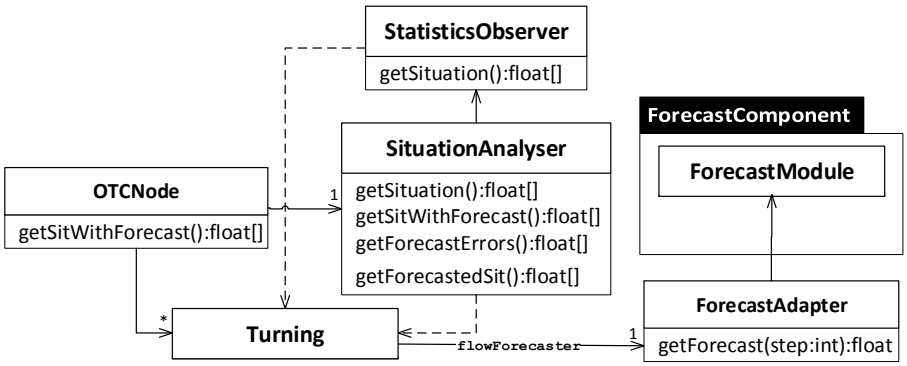


Figure 6.2: A class diagram showing the most important entities to create a situation description of the current and future traffic volumes.

6.3.2 Forecasting the traffic flow of turning movements

Analogously to vector $\vec{M}$ describing the current traffic demands, forecasts have to be derived for each of the intersection's turning movements. Therefore, each *Turning* is extended by a *ForecastAdapter* (see figure 6.2). This entity serves as an adapter between the *Turning* module and the *ForecastModule*. This adapter triggers its associated *ForecastModule* to generate forecasts of the future traffic flow based on time series representing the recently monitored traffic.

Again, the principle of ensemble forecasting is implemented (see section 3.4). First, several forecasting methods P_1 to P_n independently generate forecasts based on their individual model, and historic and recently monitored data. The forecasts are generated for the point in time t when the next signal plan change can be executed. These forecasts F_1 to F_n are then combined based on the selected combination strategy. This final forecast $\hat{F}$ is calculated based on the vector of weights $\vec{w}$ derived from this combination strategy, and the forecasts $\vec{F}$ as

$$\hat{F} = f(\vec{F}, \vec{w}) = \sum_{i=1}^{n} F_i * w_i. \tag{6.3}$$

The weights $\vec{w}$ are iteratively updated based on the forecast errors derived from the actual traffic flow volumes and the respective forecasts.

6.3.3 Weighted combination of traffic flow and forecasts

Having generated a vector $\vec{F} = (\hat{F_1}, \ldots, \hat{F_n})$ consisting of the combined forecasts for each of the intersection's turning movements, $\vec{F}$ is combined with the current traffic flow $\vec{M}$ as given by equation 6.2.

Algorithm 1: Building a situation description combining traffic flow forecasts and the current traffic situation.

Data: situation $\vec{M}$, forecasts $\vec{F}$, forecast errors $\vec{E}$
Result: adjusted situation $\vec{M_{adj}}$

```
1  M_adj ⟵ ∅
2  error_max = 1.0
3  for i = 0; i < M.length; i++ do
4  |   error = |E[i]|
5  |   if error < error_max then
6  |   |   α = error/error_max
7  |   |   M_adj[i] = α * M[i] + (1 − α) * F[i]
8  |   else
9  |   |   M_adj[i] = M[i]
10 |   end
11 end
12 return M_adj
```

Algorithm 1 depicts the combination process. Remember, vector $\vec{M}$ describes the current traffic conditions at the local intersection (one value for each turning), and vector $\vec{F}$ contains a forecast value for each turning. These two vectors are combined based on the vector of forecast errors $\vec{E}$. Each forecast error is calculated independently for each turning's forecast component in a sliding window fashion with the mean absolute scaled error (MASE) (cf. 3.5). MASE compares the absolute deviation between the current traffic flow Y_t and the traffic flow Y_{t-1} of the previous measurement. The lower the MASE value, the higher the influence of the forecasts on the final combination (see lines 5 to 7). For example, if the MASE is zero for a certain turning, the current value is neglected and only the respective forecast value is used at the according

position in M_{adj}. In case the MASE is higher than a previously defined maximal threshold, the currently monitored sensor values are taken without adaptation (line 9). Otherwise, the forecast error accounts proportionally to the final result. Finally, the adjusted situation vector M_{adj} is returned. The result is an adjusted vector of traffic flow combining the forecasts and the monitored traffic flow of each turning, according to the estimated forecast accuracy of the respective forecast module.

6.3.4 Forecast-augmented green time adaptation

With this weighted combination of the current traffic flow and the forecasts of the turning's future traffic flow, the new situation vector is passed on to the controller at layer 1. As usual, the controller selects a new signal plan matching the input vector provided by the observer which is finally activated once the currently active signal plan's cycle is finished.

6.4 Evaluation

In order to demonstrate the benefit of the forecast-augmented adaptation of signal plans, a simulation study is conducted to evaluate the impact of traffic flow forecasts within the self-adaptive traffic control strategy of OTC. Therefore, a simulation model of an urban area situated in Hamburg, Germany was developed. It reflects the actual topology, traffic conditions as derived by a census, and the actual control strategies running in reality. Furthermore, an artificial Manhattan-style network with six intersections is used to evaluate the approach in a more complex scenario. The evaluation is done with the traffic simulation toolkit *Aimsun 8.0 Professional* [BC02], which is a widely used tool in the field of professional traffic engineering. In the following, the new proactive adaptation of the signal timings (OTC-Pro) is compared against

- the standard OTC system as introduced in section 2.2,
- and against the fixed-time control strategy installed in reality.

Evaluation criteria: Each approach is evaluated based on following performance criteria:

- The average travel times for the complete network in seconds per kilometre,
- the average stop time for the complete network in seconds per kilometre,
- the mean queue of vehicles in front of red traffic lights,
- and the vehicle's emissions: nitrogen oxides NO and NO_2 (NOx), carbon dioxide (CO_2), and particulate matter (PM).

The combustion of fossil fuels results in the emission of several pollutants. The three chosen pollutants are harmful to the human health. The emission of these pollutants has been estimated with the help of Aimsun's internal microscopic pollution emission model which is based on [PBL06]. Each vehicle's emission is estimated based on its current speed and acceleration using non-linear multiple regression techniques. They are emitted especially during high load and idling periods.

6.4.1 Simulation setup

To remove random factors, the results are averaged over three independent replication runs. Each replication is based on another random seed, therefore influencing the random behaviour of the simulation, giving out different results. To simulate physical sensor input, every 0.75 seconds, the values of the simulated traffic parameters are sent from Aimsun to OTC. For turnings, the sensor values are stored and averaged over a time horizon equal to the cycle time of the currently active signal plan. Accordingly, forecasts can be derived for time intervals which are multiples of the cycle time.

Configuration of the forecast module

Choosing a set of forecast methods is quite difficult since there is a huge number of different techniques. However, the insights derived from previous chapters give us some guidelines. As traffic flow normally shows non-stationary behaviour, it is reasonable to use an $ARIMA(p, 1, q)$ model with an integrative part. In this case a non-stationary stochastic process is assumed. Furthermore, the autoregressive and moving average part should be chosen of higher orders, such as $ARIMA(2, d, 2)$ or $ARIMA(3, d, 3)$. The *forecast* package for R provides auto.arima, an implementation of ARIMA. It automatically adjusts its parameters according to several time series measures and returns the best model found. By applying auto.arima to some representative traffic flow curves, our initial thoughts are confirmed.

Finally, to evaluate this approach, three ARIMA processes with different models are used. Following the idea of ensemble forecasting, the forecasts of $ARIMA(2, 1, 2)$, $ARIMA(3, 1, 3)$, and $ARIMA(0, 1, 1)$ are combined. They create their forecasts based on the last 20 monitored traffic flow values. Of course, experiments using different numbers of recently monitored values were carried out, however 20 turned out to be the optimal value. Lower values resulted in drastically worse results, whereas higher values were slightly worse than considering the last 20 values. The *forecast* package offers the setting of confidence boundaries associated with the prediction intervals. The selected forecast methods are used as is from this package. A confidence level of $[80; 95]$ is chosen for the intervals used with ARIMA.

When choosing a combination strategy, the guidelines summarised in section 3.4.2 are considered. As no definitive assumption can be made about the performance of these models in all situations, and therefore about which one should be weighted higher than the others, the simple average [BS16] is selected as combination strategy. This strategy was favoured because of its simplicity and robustness.

As described earlier, the MASE measure is used to evaluate the forecast accuracy of each individual forecast module providing forecasts for each turning. To calculate MASE, the last ten pairs of forecasts and their actual values are considered. A new forecast is generated for the point in time when two consecutive cycles of the currently active signal plan are finished. Based on a cycle time of 90 seconds, a forecast is created every three minutes. Consequently, the first forecasts are available after about 30 minutes (20 data points measure in intervals of 90 seconds), the first accuracy estimates based on the MASE measure after about 30 more minutes (10 data points measured in intervals of 2*90 seconds).

Forecasts are only considered within the combination when they have a certain accuracy defined by the MASE measure. Forecasts are considered accurate if MASE is below a certain threshold (i.e. $MASE < 1$). As forecasts are most accurate when they have a MASE value below one, the threshold is set to one accordingly. Other thresholds did not improve the performance.

Configuration of OTC's learning components

Table 6.1 depicts the most important parameters for the XCS at layer 1 and table 6.2 shows the configuration of the evolutionary algorithm at layer 2. Most parameters are set according to findings and recommendations of [Pro11].

As described in section 2.2, XCS's task is to store mappings between traffic situations to green phase durations. The size limit of the rule base defines the maximal number of rules (classifiers) which can be stored. In case new classifiers are created by the evolutionary algorithm, classifiers with a bad rating are removed. The learning rate β defines how fast XCS adapts to new inputs. Higher values lead to faster adaptation but also result in higher variance. How good a classifier is estimated to perform is dependent on its accuracy parameters, its fitness, and prediction error.

Table 6.1: Configuration of the XCS at layer 1 used for the experiments.

Parameter	Value
Activation interval	2
Rule base size limit N	200
Learning rate β	0.2
Initial prediction error ϵ_I	25
Initial fitness ϕ_I	0.01
Accuracy parameters	$\alpha = 0.1$, $\epsilon_0 = 2$, $\nu = 5$

Table 6.2 summarises the most important parameters for the evolutionary algorithm at layer 2. The task of the genetic algorithm is to find the optimal signal plan for the given traffic condition. This is achieved by adapting the green times of the signal phases. We set the maximum cycle time at 120 seconds. The minimum cycle time depends on the intersection. The population size defines the maximal number of possible solutions that can be stored. We use the $(\mu + \lambda)$ selection (plus strategy). It is generated from the union of the parent population and the number of offspring that is created. The number of generations limits the execution duration of each run and the simulation duration defines its stopping criterion.

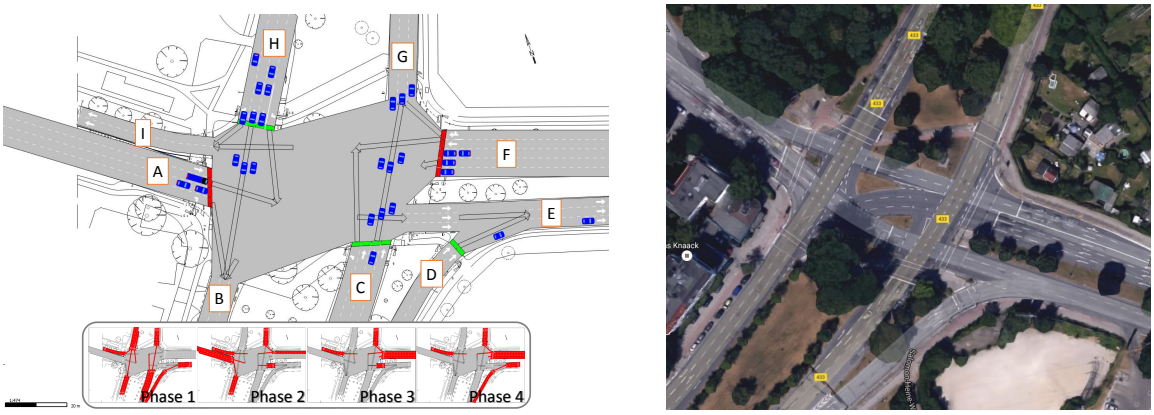
Table 6.2: Configuration of the evolutionary algorithm at layer 2 used for the experiments.

Parameter	Value
Population size	16
Number of offsprings	24
Maximal number of generations	64
Simulation duration	7200 s
Selection strategy	plus strategy

For more information on the evolutionary algorithm, the XCS, and the according evaluation and parameter study, the interested reader is referred to [Pro11].

6.4.2 Scenario I: Isolated intersection

The following simulation study uses a highly realistic simulation model of an isolated intersection located in Hamburg, Germany (Figure 6.3). K3 is a four-armed intersection with eleven turnings and part of the federal highway 433. It connects four roads: Alsterkrugchaussee (G, H), Deelböge (E, F), Rosenbrook (B, C, D), and Borsteler Chaussee (A, I). K3 has 25 signal groups, ten of which serve motorised traffic movements. Six inductive loop detectors serve as input to the traffic-actuated control. The traffic data was collected on Tuesday, May 4, 2004. In this census, cars and trucks passing the intersection were counted with a time resolution of 15 minutes. The simulation duration ranges from 5.30 a.m. to 12.30 p.m. including a typical morning rush hour scenario (7 a.m. to 10 a.m.). OTC and OTC-Pro start without further knowledge, only the installed fixed-time signalisation plan is known in advance.



(a) Aimsun's simulation model and the according green phases

(b) Aerial photo (© Google Maps, 2016)

Figure 6.3: Intersection K3 at Hamburg, Germany.

Traffic demand: Figure 6.4 illustrates the traffic profile for three different replications as simulated by Aimsun for the simulation period. The horizontal axis shows the time and the vertical axis depicts the traffic flow in vehicles per hour. The simulated period starts with low traffic density before it drastically raises up to 8000 vehicles per hour between 6.30 a.m. and 8.30 a.m. After 9 a.m. the traffic density slowly decreases to a medium level. The ratio of passing trucks compared to cars lies between 2% and 3%.

Signal plan: The simulation model depicts the real topology including the actual fixed-time signal programs that have been developed by traffic engineers. Table 6.3 shows the fixed-time signal timings as deployed in reality which is later-on used as a reference in the simulation study. The signal plan has four phases and defines the green times and the interphase durations in between green phases. Phase 1 has allocated a green time length of 33 seconds, phase 2 of

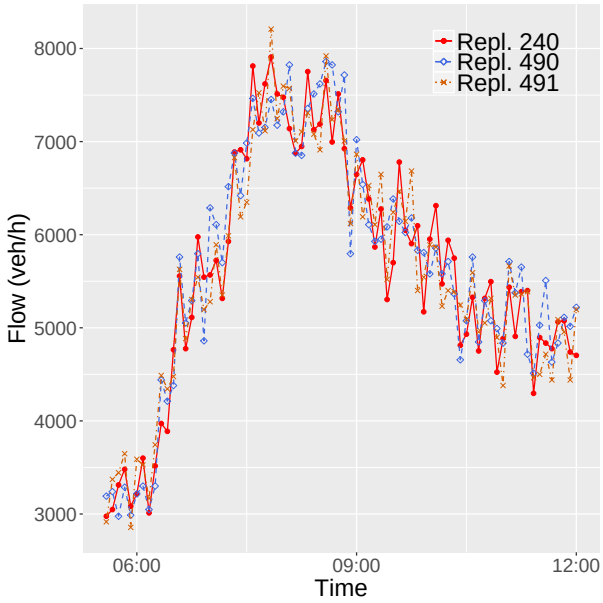


Figure 6.4: Vehicular traffic flow profiles at K3 for three different replications. The simulation ranged from 6 a.m. to 12.30 p.m., the flow is given in vehicles per hour.

nine seconds, phase 3 of three seconds, and phase 4 of nine seconds. The duration of yellow light is three seconds and the signal plan's cycle time is 90 seconds.

Table 6.3: Fixed-time signal plan for intersection K3. Signal timings are given in seconds (IP=Interphase, GT=Green time). The duration for yellow light is three seconds.

Phase 1		**Phase 2**		**Phase 3**		**Phase 4**			Cycle
IP	GT	IP	GT	IP	GT	IP	GT	IP	time
10	33	13	9	4	3	7	9	2	90

To get a better feeling for the adaptation strategy of OTC-Pro, the signal plans evolved by Layer 2 and actuated at layer 0 have to be investigated. Table 6.4 shows the development of the green timings and the according cycle times over time using OTC-Pro with forecasts. Compared to the traffic flow developments depicted in figure 6.4, a pattern can be deduced how OTC-Pro tries to optimise the signalisation. Right from the start, the initial fixed-time signal plan is changed to a plan with a much shorter cycle time of about 60 seconds. As the density of traffic raises, the assigned green times become longer and with them the cycle time. This effect can be observed between simulation minutes 85 to 160. This effect can be attributed to the fact that the traffic flow exceeds 6000 vehicles per hour during this time period. Before and after, the cycle time mostly ranges between 60 and 75 seconds.

Table 6.4: The signal plans as evolved by OTC-Pro at layer 2 using traffic flow forecasts (Replication 240). Interphases are omitted as they stay unchanged.

Time (min)	**Green times (s)**				**Cycle time (s)**
0.0	33	9	3	9	90

15.8	11	5	3	5	60
36.0	12	5	3	5	61
42.1	13	5	3	5	62
46.7	15	6	3	5	65
49.9	16	5	3	5	65
53.1	18	5	3	5	67
66.4	18	6	4	5	69
68.1	19	6	5	5	71
73.3	18	7	3	5	69
76.7	17	8	7	5	73
83.9	7	5	4	5	57
85.8	23	10	8	5	82
87.5	22	10	6	5	79
95.5	10	7	3	5	61
99.4	8	7	3	5	59
101.5	10	9	3	5	63
103.5	37	24	13	9	119
117.4	36	17	10	9	108
123.1	33	16	10	5	100
128.3	38	15	10	5	104
141.8	35	12	9	5	97
149.3	31	9	4	5	85
156.9	25	12	5	5	83
166.8	22	5	3	5	71
170.5	22	5	3	5	71
179.4	23	7	3	5	74
182.9	22	8	3	5	74
188.6	27	9	5	5	82
194.0	23	8	3	5	75
223.7	18	7	3	5	69
241.3	20	5	3	5	69
246.5	23	5	3	5	72
253.7	18	6	3	5	68
258.7	16	9	3	5	69
262.2	17	8	3	5	64
263.9	16	8	3	5	68
269.0	13	8	3	5	65
272.4	13	6	3	5	63
278.8	16	5	3	5	65
283.7	19	5	3	5	68

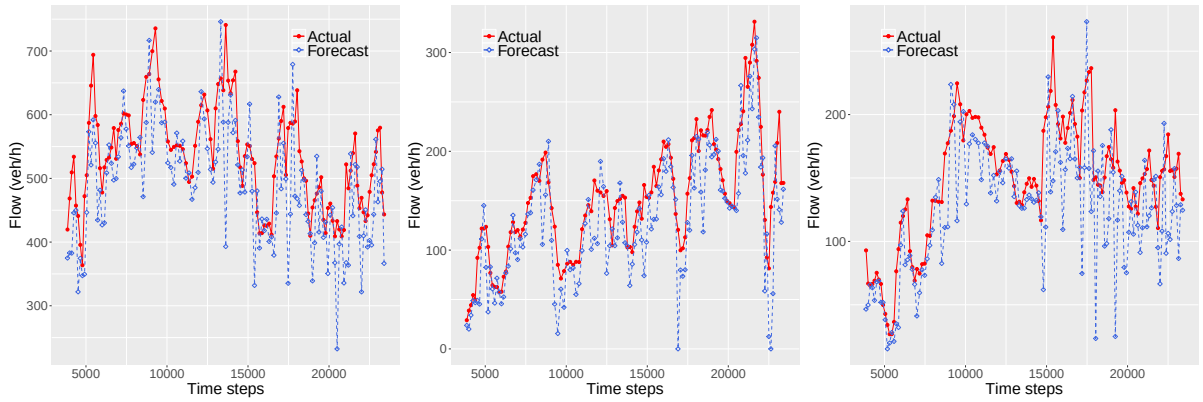
Compared to the standard OTC control, the number of optimisations triggered at layer 2 is reduced by the use of forecasts. Consequently, the final population of XCSF consists of less classifiers as well. Remember, OTC only creates new rules in case of unknown traffic conditions.

Table 6.5: Classifier population size and number of layer 2 optimisations for K3.

	OTC			OTC-Pro		
	Min.	Max.	Avg.	Min.	Max.	Avg.
Layer 2 optimisations	66	73	70.3	46	51	48.3
Final population size	107	122	113.3	85	93	89.3

Forecast accuracy

To show its performance each signal plan has to be active at least for two full cycles. Consequently, the forecast horizon equals twice the signal plan's cycle time, i.e. $2 * 90s = 180s$. Figure 6.5 shows three time plots with the actual and the forecasted traffic flow profiles for representative turnings of K3 from replication 240 (chosen from a total of 13 turnings). The horizontal axis displays the simulated time step, the vertical axis represents the vehicular flow in vehicles per hour. Although, the actual traffic flow (dotted line) has heavy fluctuations, the forecasts (solid line) are mostly rather precise. All plots show that the forecasts tend to show numbers that are too low in certain situations. Usually, this happens when traffic flow decreases.

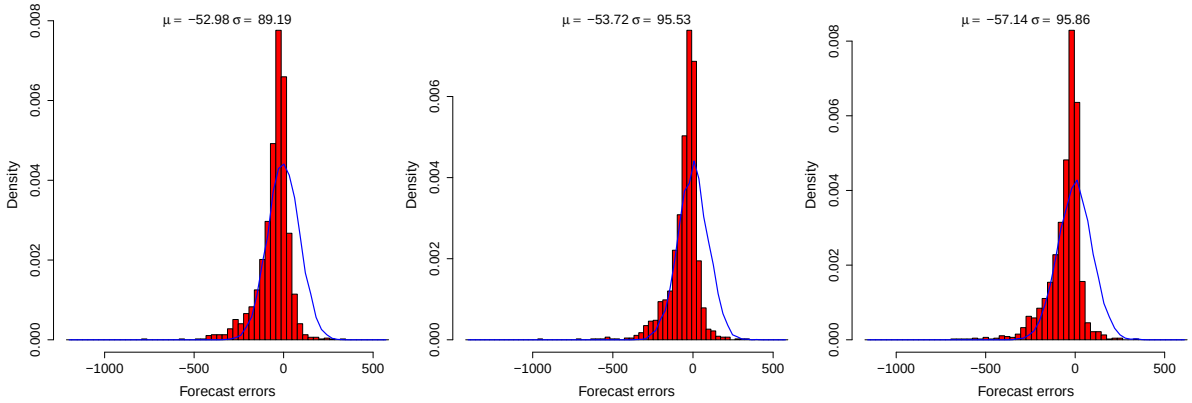


(a) Turning 1, section F to G **(b)** Turning 3, section C to G **(c)** Turning 8, section A to B

Figure 6.5: The time plots show the temporal variations between the actual and the forecasted traffic flow for representative turnings at K3 (Fig. 6.3) for replication 240.

The histograms of the forecast errors in figure 6.6 support this observation. Still, these plots show that the distribution of forecast errors is roughly centred around zero, more or less normally distributed, and slightly skewed to the left compared to a normal curve (solid line). The absolute forecast errors range between -693 and +339 with a standard distribution of 94.7 and a mean forecast error of about -57.8 vehicles per hour. Compared to the average flow per hour of around 6000 vehicles, the forecast error is low.

The majority of the MASE values range between 0.8 to 1.9. A few times, temporarily higher values are observed for some turnings. Usually, the best MASE value within the simulations did not go below a value of 0.7. We use algorithm 1 with a threshold of 1.0. With this setup, forecasts contribute at most one-third to the combined result. This conservative approach is chosen since suboptimal signal timings can rapidly lead to congestion. At last, the experiments



(a) Replication 240 **(b)** Replication 490 **(c)** Replication 491

Figure 6.6: Density histograms showing the absolute forecast error distribution for three different replications of K3 compared to a normal distribution curve.

show that the average MASE did not correlate with the average traffic flow at a turning (i.e. higher traffic flow did not result in higher forecasts errors).

Simulation results

The final results of the simulation runs are presented in table 6.6. Compared to the standard OTC (fixed-time signal plan), the forecast-augmented OTC-Pro control significantly reduces the mean queue by 6.3% (18.2%) and the average stop time by 5.3% (15.8%). Consequently, OTC-Pro reduces greenhouse gas emissions by vehicles, such as CO_2, (FTC: 2.2%, OTC: 1.1%) and NOx (3.4%, 1.5%), but has slightly higher values for PM (-1.6%, 0.3%).

Table 6.6: Simulation results for K3 for three replications. Best results are highlighted in bold.

	FTC			**OTC**			**OTC-Pro**		
	240	490	491	240	490	491	240	490	491
Mean queue [veh]	49.0	54.7	53.7	45.1	48.4	**45.6**	**43.1**	**43.7**	46.9
Stop time [s/km]	143.5	157.6	156.3	133.9	141.2	**136.0**	**130.5**	**128.8**	141.1
Delay [s/km]	162.0	177.4	175.9	152.8	160.9	**155.4**	**150.0**	**148.2**	161.0
CO_2 [g/veh]	553.7	572.3	570.1	**554.0**	567.2	**559.8**	554.7	**556.5**	565.2
PM [g/veh]	**0.325**	**0.333**	**0.336**	0.339	0.339	0.338	0.337	0.336	0.341
NOx [g/veh]	1.57	1.62	1.62	**1.56**	1.59	**1.58**	**1.56**	**1.55**	1.59

Figure 6.7 shows the time series for the total average delay in seconds per kilometre. The depicted results are averaged over three simulation runs with different random seeds. Error bars are omitted for improved readability. Using the static fixed-time signal plans, vehicles are faced with an average delay of 171.7 seconds per kilometre. The use of OTC-adapted signal plans reduces this value by 9.0% to 156.4 seconds. Compared to the standard OTC (fixed-time signal plan), the additional consideration of forecasts results in further improvement by 2.1%

(10.9%), reducing the average delay to 153.0 seconds. OTC-Pro flattens the steeply rising peak in the morning that occurs from 6.30 a.m. to 8 a.m. before developing to a normal level until 10 a.m. The time until a normal level of delay is reached, can be reduced. By reducing the average delay, the traffic flow is smoothed and repeated acceleration and deceleration can be avoided.

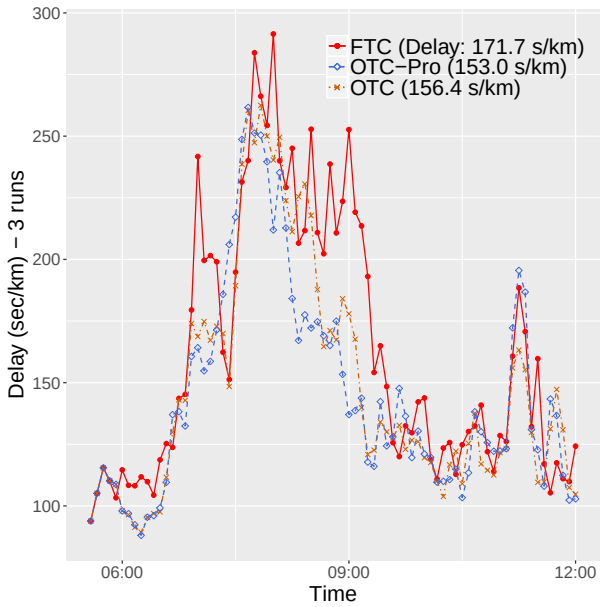


Figure 6.7: Mean delay for K3 averaged over three simulation runs. Compared to the reference solution, OTC-Pro reduces the average delay.

Finally, we investigate what effect using forecasts without considering forecast accuracies or the actually monitored values has. To conclude, this approach turns out to be worse than the standard forecast-augmented OTC-Pro control. However, with an average delay of 154.1 seconds per kilometre, an average stop time of 134.7 seconds per kilometre, and a mean queue of 44.9 vehicles, its performance is still better than the standard OTC (delay: 156.4 s/km, stop time: 137.0 s/km, queue: 46.4 vehicles).

Table 6.7: Averaged simulation results for OTC-Pro, creating the situation description but only considering forecasts.

Measure	**240**	**490**	**491**	**Average**
Mean queue [veh]	42.7	43.3	47.8	44.5
Stop time [s/km]	129.7	128.8	143.9	134.1
Delay [s/km]	149.0	148.0	164.1	153.7
CO_2 [g/veh]	554.5	554.3	566.9	558.5
PM [g/veh]	0.339	0.335	0.344	0.339
NOx [g/veh]	1.56	1.55	1.60	1.57

6.4.3 Scenario II: Manhattan-style road network

The results of the previous experiment for an isolated intersection are quite promising. However, the approach has yet to be evaluated for more complex scenarios. For this, an artificial simulation model of an urban Manhattan-style road network was developed with six intersections that are located in two rows of three intersections each (Figure 6.8). The intersections have an identical topology that allows for all possible turning movements. Each of the six intersections supports twelve turning movements. The connecting two-laned road segments have a length of 250 m to 350 m, and provide an additional side-lane for left-turns. The simulation duration is three hours and 20 minutes. The configurations of OTC and of the forecast module are the same as in the previous experiment.

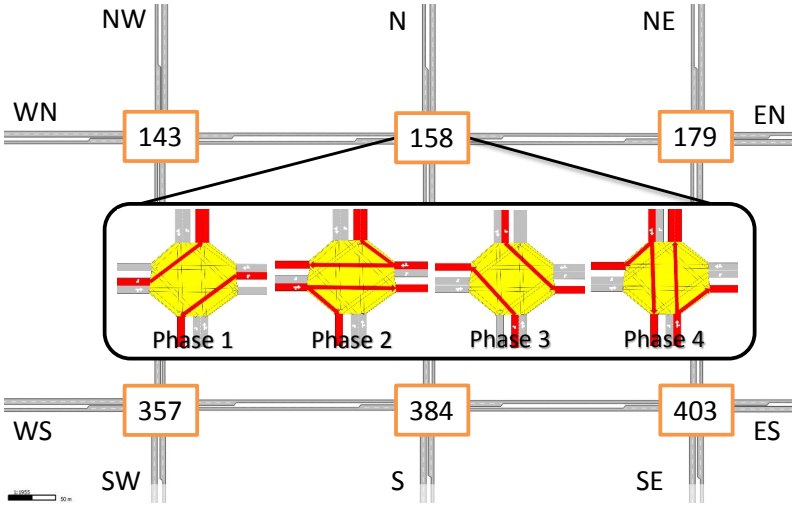


Figure 6.8: Artificial Manhattan-style road network with six signalised intersections. Each intersections is equipped with the same signal plan consisting of four phases with a cycle time of 90 seconds.

Traffic demand: In the simulation study, the network has been simulated for three hours and 20 minutes. Table 6.8 summarises the simulated demand for the Manhattan network. The origin/destination pairs describe the number of vehicles per hour traversing the network from one border to another. In the first twenty minutes, a warm-up phase with low traffic density is simulated. Afterwards, the demand increases drastically. The east- and westbound streams are the most heavily travelled. During the third phase, the overall traffic load decreases again and the strongest streams are changed to the north- and southbound streams. These changes in demand resemble the typical phenomena of traffic, where major travel directions are changing in the course of a day.

Signal plans: Each of the Manhattan networks intersection's fixed-time signal plans consists of four phases (see figure 6.8). The according signal timings are presented in table 6.9. Phases 1 and 3 serve vehicles turning left, while Phases 2 and 4 serve vehicles going straight or turning right. Phase 1 has a green time of 10 seconds, Phase 2 has a green time of 23 seconds, and so on.

Table 6.8: Traffic demands for the Manhattan network as origin/destination (O/D) pairs.

O/D pairs (in veh/h)			0:00 - 0:20	0:20 - 1:50	1:50 - 3:20
WN	→	EN	120	600	200
EN	→	WN	120	600	200
WS	→	ES	60	300	200
ES	→	WS	120	600	400
NW	→	SW	60	300	200
SW	→	NW	60	300	400
N	→	S	60	300	200
S	→	N	60	300	400
NE	→	SE	60	300	200
SE	→	NE	60	300	400
Others			180	900	600
Total			960	4800	3400

The interphases in between have a duration of five, respectively seven seconds. The duration for yellow light is three seconds. In total, the cycle time is 90 seconds.

Table 6.9: Initial fixed-time signal plan for the Manhattan network. Signal timings are given in seconds (IP=Interphase, GT=Green time). The duration for yellow light is three seconds.

Phase 1		Phase 2		Phase 3		Phase 4		Cycle
GT	IP	GT	IP	GT	IP	GT	IP	time
10	7	23	5	10	7	23	5	90

To evaluate the individual adaptation strategies of OTC and OTC-Pro, a deeper look into the signal plans created is taken. Table 6.10 summarises the minimal and maximal assigned cycle times for each executed simulation replication with OTC and OTC-Pro. Compared to the isolated intersection K3, it can be seen that the adaptation process is not as extreme. Most of the time, the average cycle time ranges between 80 and 85 seconds while the maximal cycle time does not exceed 115 seconds. During the first two hours, OTC usually assigns signal plans with cycle times above 80 seconds. Afterwards, the network demand decreases and OTC reacts with cycle time reductions down to 70 seconds. Remember, FTC sticks to its fixed-time signal plan with a static cycle time of 90 seconds independent of the momentary traffic conditions.

Using the standard OTC (OTC-Pro), the average population size of each individual intersection's XCSF ranges from 32 to 46 (26 to 47), the average number of layer 2 executions lies between 17 and 24 (14 to 25).

Forecast accuracy

The histograms in figure 6.9 show the distribution of the forecast errors (the absolute deviation between the actual hourly traffic flow and the according forecast) of the executed replications.

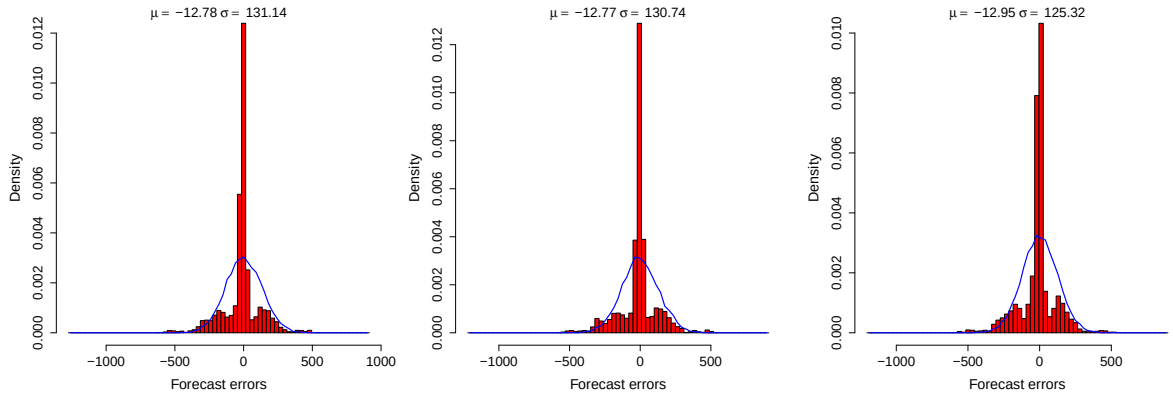
Table 6.10: The minimal and maximal cycle times as developed by OTC and OTC-Pro for each intersection and replication of the Manhattan network.

Inters.	Repl.	OTC			OTC-Pro		
		Min	Max	∅	Min	Max	∅
143	246	80	96	82.2	71	97	84.2
	323	71	104	82.8	71	101	83.6
	324	73	97	80.6	69	103	80.6
158	246	75	109	88.1	70	101	85.0
	323	73	101	85.4	73	103	84.5
	324	72	102	86.2	73	105	85.2
179	246	71	109	84.3	71	105	84.9
	323	69	101	82.9	69	101	83.4
	324	67	101	82.6	67	101	81.0
357	246	75	109	87.5	71	115	86.5
	323	72	102	86.8	72	101	84.8
	324	67	108	84.0	69	100	85.8
384	246	71	100	83.4	75	109	84.4
	323	68	94	83.1	72	100	83.0
	324	69	113	84.2	70	107	83.4
403	246	70	103	85.8	73	103	85.2
	323	71	100	81.8	68	97	80.7
	324	67	105	80.9	64	101	80.7

As the histograms depict, the errors are roughly normally distributed. The mean absolute forecast error has a value of -8.1 vehicles per hour with a standard deviation of $\sigma = 134.2$ and is rather close to zero. Compared to the normal curve, it is plausible that the forecast errors are normally distributed with mean zero. However, for turnings with stronger traffic demands, the absolute error can be higher. The two peaks of the distribution (one to the left and one to the right) are attributed to this observation.

Simulation results

Again, three independent replications with different simulation seeds were executed. The average results obtained for the Manhattan network are depicted in table 6.11. In contrast to the results from the previous experiment with an isolated intersection, the benefit of the forecast-augmented OTC-Pro control is less high. However, OTC-Pro can slightly reduce the average delay (see figure 6.10 and table 6.11). A decrease in the number of stops consequently leads to lower pollution emission. Compared to OTC, OTC-Pro reduces the average travel time by 1.1% and the average stop time by 1.7%. Both OTC and OTC-Pro can drastically reduce the network-wide pollution emission compared to fixed-time signal control. FTC results in much higher pollution rates, as the internal model of Aimsun estimates higher emission during congested and start-stop conditions. The average results are summarised in table 6.12.



(a) Replication 246 **(b)** Replication 323 **(c)** Replication 324

Figure 6.9: Manhattan network: Histograms showing the absolute forecast error distribution for three replications compared to a normal distribution curve.

Table 6.11: Simulation results for the Manhattan network for three replications. Best results highlighted in bold.

	FTC			**OTC**			**OTC-Pro**		
	246	323	324	246	323	324	246	323	324
Travel time [s/veh]	691.3	646.0	790.8	466.7	**460.0**	461.7	**455.9**	460.5	**456.2**
Stop time [s/km]	156.3	144.3	189.6	87.4	**85.5**	85.5	**84.0**	85.8	**84.3**
Delay [s/km]	170.9	157.7	205.3	98.4	**96.5**	96.5	**95.0**	96.8	**95.2**
CO_2 [g/veh]	1042.6	1001.9	1096.1	94.4	94.1	94.4	**94.0**	**94.0**	**94.0**
PM [g/veh]	0.508	0.493	0.535	**0.045**	**0.046**	**0.046**	0.046	**0.046**	**0.046**
NOx [g/veh]	3.06	2.99	3.29	**0.27**	**0.27**	**0.27**	**0.27**	**0.27**	**0.27**

Next, it was investigated if larger sets of forecast methods can improve these results. A set of four (ARIMA(0,1,3) added) and a set of five methods (ARIMA(0,1,3) and ARIMA(0,1,1) added) is evaluated. To conclude, the results were rather similar to the previous findings. Larger set sizes did not result in any noteworthy improvements. Also making forecasts based on shorter or longer time series (the last 10, 15 or 25 values instead of 20) did not lead to better results.

The time series in figure 6.10 show the network-wide delay averaged over three independent replications. Error bars are omitted for improved readability. After an initial warm-up time of 15 minutes, OTC starts to create new signal plans, outperforming the reference solution right away. The static fixed-time signalisation results in congested conditions throughout almost the whole simulation period. After 15 more minutes, the forecast methods have received enough data to generate forecasts. In the first two hours, OTC and OTC-Pro perform on similar levels, OTC-Pro having a slightly lower delay. After two hours, the traffic demand decreases and the major streams change now going from north to south and vice-versa. OTC-Pro clearly outperforms OTC from thereon. Considering the complete simulation period, the fixed-time control has a total average delay of 172.7 seconds (standard deviation $\sigma = 91.3s$). OTC reduces this value by 42.4% to 99.4 seconds ($\sigma = 51.7s$). The additional consideration of forecasts further reduces

Table 6.12: Mean simulation results for the Manhattan network averaged over three replications.

	FTC	OTC	OTC-Pro
Travel time [s/veh]	709.4	462.8	457.5
Stop time [s/km]	163.4	86.1	84.7
CO_2 [g/veh]	1046.0	94.3	94.0
PM [g/veh]	0.521	0.046	0.046
NOx [g/veh]	3.11	0.27	0.27

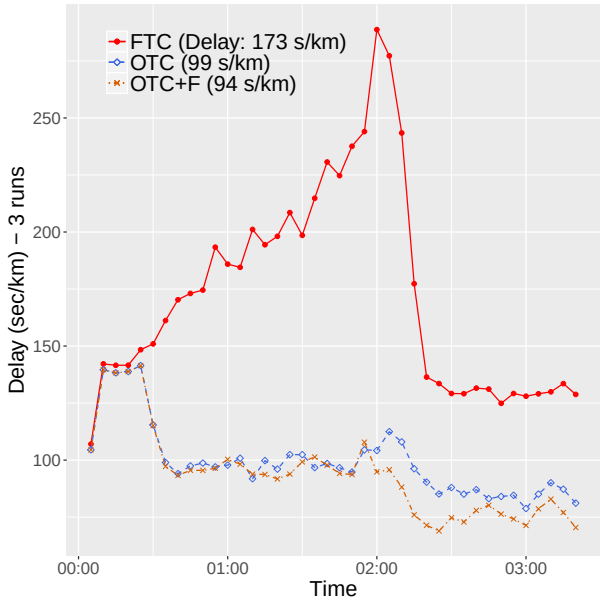
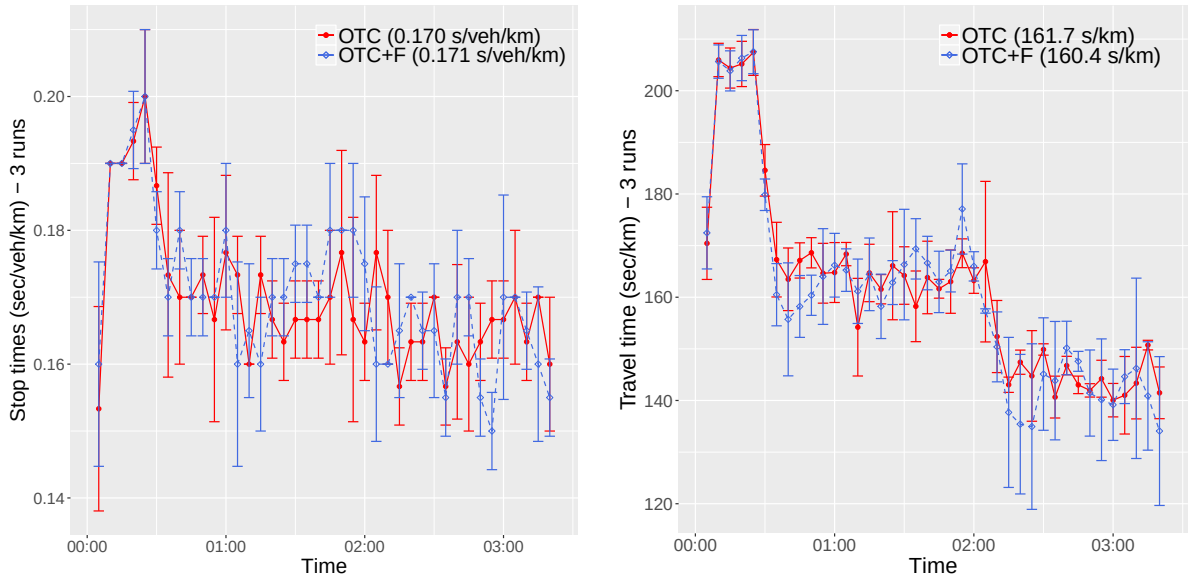


Figure 6.10: Mean delay in seconds per kilometre for the Manhattan network averaged over three simulation runs.

the average delay to about 93.8 seconds ($\sigma = 47.1s$) (improvement of 45.7% compared to FTC, 5.6% compared to OTC).

Figure 6.11 depicts the travel times and stop times monitored over the simulation period for OTC and OTC-Pro. The results are averaged over three independent replications, each one simulated with another random seed. Additional error bars represent the standard deviations. In the beginning, travel times increase as more and more vehicles enter the network. After the initial warm-up time, OTC populates its initial empty rule base and reduces the average travel time. Interestingly, during both periods of travel time reduction (30 minutes and 120 minutes), OTC-Pro results in faster and stronger reductions compared to OTC. However, the average stop time is almost the same for both approaches.



(a) Stop times

(b) Travel times

Figure 6.11: Travel times and stop times for the Manhattan network.

6.5 Summary

This chapter presented how concepts of time series forecasting can be integrated beneficially into a self-adaptive and self-organising traffic management system, such as the distributed OTC system. Its benefit for the traffic signal adaptation was evaluated on the basis of a simulation-based evaluation.

It can be concluded that the use of forecasts can lead to an improved traffic signalisation. This anticipatory approach was compared to the standard OTC system and to a real-world controller, working on fixed-time signal plans that are actually deployed in Hamburg, Germany. Using this setup, it was demonstrated that the proactive adaptation strategy outperforms the standard OTC and the reference solution, especially in highly demanding conditions such as rush-hours. This is especially promising since peak load poses the most severe challenge to traffic control. The anticipatory approach based on forecasts of traffic flow leads to a better control strategy, i.e. in terms of decreased travel times and pollution emission. Interestingly, the use of forecasts did not only improve important factors, such as the average delay and throughput at signalised intersection, but also lead to smaller population sizes within the learning component of OTC. To conclude, the forecasts-augmented adaptation strategy of OTC-Pro is especially beneficial during high-volume traffic. When traffic density is low, the standard OTC system suffices to optimise the signalisation.

Chapter 7

Anticipatory and adaptive traffic guidance

Congestion is a serious problem affecting all traffic participants. The resulting waste of time and fuel leads to billions of dollars lost annually [SEL15]. Urban road networks come to their capacity limit due to increasing demands. The complexity of these road networks, due to mutual influence of different traffic control strategies, is no longer feasible for a central instance. Thus, decentralised, self-organising and self-optimising approaches that better utilise existing infrastructure by optimised vehicle coordination, are needed. OTC represents a decentralised, self-organised traffic management system. Using the communication infrastructure, the local delays occurring at signalised intersections and the estimated travel times based on the current traffic streams within the road network can be communicated to other nearby intersections. Based on this disseminated information, OTC-controlled intersection controllers have the ability to determine the shortest paths to prominent destinations (such as the main hall or the main station). Consequently, the benefits of dynamic route guidance (DRG) are the alleviation of congestion, the enhancement of the reliability and robustness of the road network, and the provision of navigational assistance for travellers which are unfamiliar with the network [Don11].

In general, routing protocols compute the fastest or shortest route from a starting point to a destination. This calculation is typically based on static information about the road network, or is dynamically calculated based on recently monitored traffic data. The computed routes can be disseminated via variable message signs or navigational systems, and give drivers an indication, how to traverse the network to reach their destination as fast as possible. However, this reactive approach yields significant problems: First, the drivers need time to follow their proposed route. Therefore, the initial route recommendation might already be outdated at the next intersection. This leads to repeated re-routings, that will decrease the acceptance of the route guidance mechanism. Second, especially in situations with developing congestion, a fast reaction is valuable to avoid further negative impacts on traffic.

In contrast, a proactive routing protocol takes into account forecasts about future traffic flow. By considering both the current traffic conditions and the estimated traffic flow forecasts, the existing reactive DRG system is turned into a more robust, proactive, and anticipatory one. Forecasts about future traffic flow patterns help to detect capacity shortages in advance. Furthermore, this approach has the potential to reduce the re-routing demands, leading to a broader acceptance of the system, and making it more reliable [STH15b, Fu01].

This chapter presents two novel, anticipatory and time-dependent route guidance protocols: Temporal link state routing (TLSR) and temporal distance vector routing (TDVR). These protocols are used to distribute knowledge about forecasts of the traffic flow for several future time steps in combination with current route states. Thereby, these concepts extend two well-known

Internet protocols, the distance vector routing (DVR), respectively the link state routing (LSR) protocol. These protocols were adapted for urban road networks in previous work [PTL+12], but just considering the current traffic conditions. Therefore, these protocols only reacted to real-time data without any insight into future conditions. In a first step, the route guidance module is coupled with the forecast module of the observer. Thereby, the intersection controllers are no longer limited to react on historic traffic measurements, but to incorporate short-term forecasts of future traffic developments. The major goals of the novel protocols are the improvement of the network's robustness, the minimisation of the average travel time, and the reduction of emission by preventing congestion. A better distribution of traffic streams makes it possible to use the capacity of the road network more efficiently. The performance of these dynamic anticipatory protocols is investigated in a simulation-based evaluation within OTC with two artificial networks under free flow and disturbed conditions.

7.1 Problem formulation

> "The shortest distance between two points is under construction."
>
> Noelie Altito

Dynamic networks are often used as one representation for network-structured decision-making problems over time [Aro89]. Many real world cases that could benefit from optimisation, such as energy systems, communication systems, and traffic networks can be mapped to graphs and can therefore be seen as one representation of the underlying network structure. Thus, well-established graph algorithms can be used to solve these problems. Many existing approaches make the simplifying assumption that the weights of the network's arcs (representing costs) are constant over time. However, real-world systems are usually time-dependant with regard to the time-varying costs of the arcs [DBKS10].

The route planning problem can be formulated as the computation of an optimal point-to-point shortest path in a road network [NBB+08]. In this scenario, optimality refers to the path with the shortest distance or with the minimal travel time. The calculation of these routes can rely on static information only, but can also incorporate real-time information or forecasts of future traffic developments. If only static information without knowledge of the current traffic situation is used, the resulting graph is a static representation of the road network. The domain of finding the shortest paths in graphs is an entire field of research for itself. *Delling et al.* [DSSW09] present an extensive overview, focussing on algorithms applied to road networks.

In the following, several important definitions are introduced, finally leading to the definition of a time-dependent network.

Definition (Static graph) *Let $G = (V, A)$ be a directed graph with a source node $s \in V$ and a target node $t \in V$. Each arc $a \in A$ has an associated positive capacity c_a, and a transit time $\tau_a \geq 0$. The capacity c_a equals the maximum number of vehicles that can simultaneously traverse arc a. For*

arc (V_i, V_j), node V_i is called its tail, whereas V_j is its head.

A graph labelled with its assigned capacities and other additional parameters is also called *network*. In general, networks can have multiple sources (start) and sinks (destination). Yet, such networks can easily be transformed into a representation with only one source and one sink by adding a super-source node and a super-sink node. The super-source is then connected with new arcs to all sources and new arcs are also introduced from all sinks to the super-sink having infinite capacity. In the following, only *single-commodity* networks are considered, for example networks in which cars are the only commodity of interest to us (as it is sort of the case when studying traffic flow). Some authors study algorithms for *multi-commodity* flow, see for example Hall et al. [HHS03].

Several classic algorithms and their modern versions, such as the Dijkstra algorithm [Dij59], the A* algorithm [NDSL12], or the Bellman-Ford algorithm [Bel58] are applied to find the shortest s-t-path [PTL$^+$12] in static networks [KLS02]. The short-coming of these classical approaches is that they do not represent the time-varying weights of the arcs. As traffic includes a temporal dimension, several researchers presented approaches to transfer this dynamic property to time-dependent shortest path algorithms [KLS02, DBkS11]. A time-dependent graph is defined as follows [NDSL12].

Definition (Time-dependent graph) *Let $G = (V, A, c)$ be a weighted, directed graph with a source node $s \in V$, a sink node $t \in V$, an interval of time instants θ, a departure time $t_0 \in \theta$, and a time-dependent transit time function $c : A \times \theta \to \mathbb{R}_+$.*

In the setting of time-dependent networks, *time* is introduced as an additional parameter. These networks consider time-varying attributes, such as travel time, the arcs capacities, and flow costs [Bre13]. A positive *transit time* $\tau_a > 0$ is assigned to every arc $a \in A$. This represents the time it takes one unit of flow to get from an arcs tail to its head. In opposition to static networks where the transit time is constant over time, the result of the time-dependent transit time function is variable. Given a time-dependent graph with time-varying weights on its arcs, the time-dependent shortest path can be formulated as follows [DBKS10].

Definition (Time-dependent shortest path (TDSP)) *Given a graph $G = (V, A, \theta)$, a source node s, a departure time t_s at node s and a target node t, the time-dependent shortest path $TDSP(s, t, t_s)$ is the path with the minimum travel time among all paths from s to t at time instant t_s.*

In other words, our goal is to find a path $p = (s, v_1, \ldots, v_k, t)$ in G that minimises its time-dependent cost $\gamma_{t_0}(p)$. In order to model the passing of time, one can either use a *discrete* or a *continuous* setting. In the discrete setting, the time domain θ is a set of points in time $\theta = 0, 1, 2, \ldots, T$. In the continuous setting, it is an interval of real numbers $\theta = [0, T]$. Time-dependant network flow problems in real application are usually continuous processes. In order to solve continuous models, they can be converted into a discrete model. The shorter the discrete time intervals, the more complex becomes the computation. In time-dependent flow models, the

attributes of the arcs usually change in time. The function $u_a : \theta \rightarrow \Re^+$ denotes the capacity, $c_a : \theta \rightarrow \Re^+$ denotes the cost, and $\tau_a : \theta \rightarrow \Re^+$ denotes the traversal time of an arc a at a certain point in time.

Definition (Dynamic network) *Let $N = (G, u, \tau, s, t)$ a dynamic network consisting of a directed graph $G = (V, A)$ with a source node s and a target node t. The capacity function $u : A \rightarrow \Re^+$ defines the maximal amount of flow on an arc at each instant of time. The time-dependent transit time function $\tau : A \rightarrow \Re^+$ represents the amount of time that is needed to traverse an arc from its head to its tail.*

Consequently, road traffic networks can be represented by a dynamic network, whereas roads are mapped to arcs, intersections are mapped to nodes, and weights correspond to the estimated travel times. Naturally, traffic exhibits changing patterns over time, for example the arc's transit time τ is not constant but depends on the amount of flow [BK04]. A higher flow in an arc often implies a considerable increase in its transit time. Amongst other options, the time-dependent property of this transit time function can be modelled inflow-dependant [KLS02] or load-dependant [KS03]. In general, the current shortest path is not necessarily equal to the time-dependent shortest path. Concluding, a network representing the time-dependent properties of traffic is a more fitting representation then a static network model.

While there are many efficient algorithms for static flow networks, *time-dependent* or *dynamic networks* are more difficult to compute. However, if one wants to model real world problems as accurately as possible, the element of time needs to be introduced into the model.

Note on terminology: some authors prefer the term *time-dependent network* over *dynamic network*. In accordance, the former definition will be used throughout this chapter.

7.2 Related work

Research in the field of vehicular route guidance can be categorised in a variety of ways [SJ06]: static and dynamic, centralised and decentralised, and proactive route guidance. The following section gives an overview over several approaches towards route guidance and draws comparisons to OTC's self-organised and anticipatory DRG mechanism.

7.2.1 Static, reactive, and predictive route guidance

Route guidance is an important aspect in intelligent transportation systems. It provides drivers with routing proposals. These proposals can be dynamic and incorporate current and forecasts of future traffic conditions, or rely on static information only. In the following, the state of the art in the field of DRG is reviewed and the benefits and drawbacks of these approaches are discussed.

First, **static methods** compute a fixed route before the start of a trip. Today, many cars come equipped with on-board GPS-based navigation systems (such as *Garmin* or *TomTom*) [Kap05]. The navigation relies on installed maps of the network which are mostly used by modified

versions of Dijkstra's algorithm [Dij59], or the A* algorithm [HNR68]. They are applied to find the shortest route for a given starting point and a fixed destination [PTL+12, NDSL12]. *Dong* [Don11] presents a brief summary of the published literature on in-vehicle route guidance systems until 2010. He points out that static routes usually do not find the optimal route since they are not able to respect the changing traffic conditions.

Second, several **reactive routing protocols** have been proposed [Fu01, WFZ04, PST+11]. They also compute one possible route upfront, but this route can be subject to change during the trip, depending on the current position of the vehicle and the real-time information received. These protocols react to changing traffic conditions and compute a new route on-demand based on these information.

Third, **predictive routing protocols** go one step further. Apart from reacting to the current situation, they generate forecasts of upcoming traffic conditions and incorporate those forecasts into the computation of the proposed routes. It has been shown that reactive protocols and especially predictive approaches lower the average travel time in urban road networks [Fu01], whereas reactive protocols are less complex than predictive systems, at the cost of decreased robustness against incidents and congestion [SJ06].
A good first introduction to anticipatory route guidance in general is given by *Chen et al.* [CU91]. They present centralised and decentralised architectures for real-world application of anticipatory DRG and give delimitations between static and dynamic methods. *SAVaNT* (Simulation of Anticipatory Network Vehicle Traffic) [WKS00] is such a decentralised, proactive route guidance system where equipped vehicles are routed on a next-hop basis. To determine the route proposals, time-dependent link travel times are used in a traffic simulation. However, they assume that each vehicle has a compliance rate of 100% (i.e. strictly following the proposed route), and that non-equipped vehicles statically follow the shortest free-flow routes. *Claes et al.* [CHW11] present a decentralised multi-agent system based on ant behaviour for anticipatory vehicle routing. Each vehicle is equipped with a smart device, depicting an agent monitoring the vehicle's current state and location. This data is transmitted to infrastructure agents along the road and to nearby vehicle agents. Other vehicle agents use this data to forecast road occupancies to determine the best route to their destination.

7.2.2 Centralised and decentralised route guidance

DRG systems can also be classified into centralised and decentralised approaches. The former exchanges information between vehicles or controllers and a central information centre, while the latter is based on estimated link travel times and operated on vehicle unit level without central control [Don11]. A study of decentralised strategies for route guidance [FZvZ06] compared centralised and decentralised systems. The authors point out that decentralised approaches have lower computational complexity, are easily scaled and extended, while being more robust against failures and measurement errors than centralised systems, but might only come up with a suboptimal system-wide solution.
Recent decentralised approaches [WKS00, WLvB+07, CHW11] often rely on equipped vehicles with car-to-car or car-to-infrastructure communication based on floating car data (FCD). In a centralised system, all data is gathered in a traffic management centre where route proposals are computed for the entire network, taking into account system-wide objectives.
Dong et al. [DML06] propose a user-equilibrium time-dependent traffic assignment algorithm,

using the simulation assignment model DYNASMART [JMH94]. They estimate the current traffic conditions and derive short-term road utilisations to forecast travel times within a network-wide traffic simulation. However, the simulated network is rather simple and the simulated traffic behaviour is simplified. *Pan et al.* [PPZB13] present an approach which relies on intelligent vehicles that act as mobile sensors to collect real-time traffic data. Vehicles have to be equipped with GPS running a software on an embedded vehicular system or smart phone. The vehicles change their path in response to newly received guidance sent by a centralised service. The predicted fastest routes are calculated by a traffic re-routing strategy.

7.2.3 Distributed route guidance in the OTC system

In comparison to the state of the art presented in section 7.2.2 and section 7.2, the decentralised DRG mechanism in OTC poses several advantages. It is not fixed to be used with one specific routing protocol, but can be used with any protocol of the three previously mentioned categories. Depending on the protocol type, recent sensor values and additional forecasts of future traffic conditions can be dynamically incorporated.

Furthermore, the implementation is fully distributed, meaning that there is no central server collecting all data and then executing the route calculations. It thereby eliminates the single point of failure of a central server with a decentralised approach. In contrast, each controller only operates on the locally monitored data and the information received from adjacent controllers. Collaboration amongst controllers is a vital benefit of this approach. The decentralised system is easily scaled and expanded, being more robust than centralised solutions while lowering the cost for hardware components.

In contrast to some of the previous approaches, OTC does not demand additional hardware in vehicles. The traffic flow and the signalisation data are available at the responsible traffic light controller (via loop detectors or video cameras) which leads to more up-to-date information, better representing the network's current traffic conditions. Furthermore, it reduces the amount of communication. OTC provides new route recommendations at each intersection on a next-hop basis. It is assumed that not all road users follow these suggestions as each individual driver optimises his route without respect to the network-wide optimum. Previous research reports a widely varying acceptance of VMS-based route recommendations, ranging from 20% [ESH07] to 70% [ENR96]. Thus, decentralised route guidance may result in an user-optimised equilibrium and not in the system-wide optimum. This approach showed to be especially profitable during disturbed conditions [PTL+12], lowering the network-wide travel times and the number of stops.

7.3 Self-organised distributed route guidance in urban road networks

To turn OTC in an even more robust and sustainable traffic control system, a self-organised, eco-friendly route guidance mechanism has been integrated [PTB+11]. It computes the fastest routes to prominent places through the network based on the current traffic demands. Each

traffic light controller (TLC) is extended by a routing component (RC) that allows for a self-organised, fully decentralised route guidance (Figure 7.1). Communication links between neighbouring intersections allow TLCs to exchange estimated travel times and delays. These approximated travel times are then used to calculate alternative routes to prominent destinations under the current traffic demand and signalisation. With the help of a routing protocol, travel times are distributed in the network and routing tables containing the proposed routes are managed. A route is defined by an origin, a destination, and the connecting roads in-between. Each RC manages its own routing tables for each incoming section. Its entries are of the form: "to destination X, turn right, estimated arrival time: y seconds". Every signalised intersection is assumed to be equipped with VMS at their approaching sections where the route recommendations are then displayed to drivers. A vital point in this approach is the accurate estimation of travel times. The following section explains their traffic-dependent approximation within OTC.

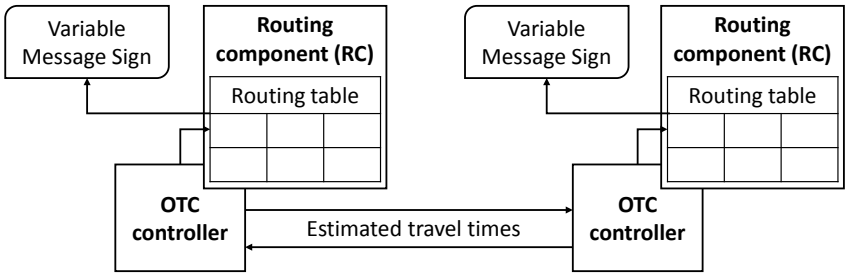


Figure 7.1: Dissemination of estimated travel times between OTC controllers for distributed route guidance.

7.3.1 Requirements for a real-world deployment

Before the routing component of OTC can be deployed in a real-world scenario, some preconditions have to be met.

1. Sensors to monitor the current traffic conditions are needed. Inductive loop detectors resemble a cheap and well established monitoring technique in traffic management applications [PX06]. Of course, other technology, such as closed-circuit television cameras (CCTV) or several types of sensors (acoustic, infrared, magnetic) can be used as well.
2. With the help of communication infrastructure, travel costs can be distributed in the network.
3. A shortest path algorithm is necessary to approximate the best routes with the lowest travel time through the network.
4. Routing protocols are needed to derive and distribute the route recommendations.
5. The calculated routes have to be provided to the traffic participants. The routing information can be passed on to the road users via VMSs (collective systems), or in case car-to-infrastructure communication is available, via direct communication to smartphones or navigational devices (individual systems).

Besides these points, OTC does not need further changes – especially no sophisticated detection and analysis devices.

7.3.2 Traffic-dependent estimation of travel times

OTC offers a collaboration mechanism between TLCs for communicational purposes. TLCs use the existing communication infrastructure to exchange their locally monitored traffic states with nearby TLCs. This situation description contains the turning delays and the estimated travel times for outgoing sections. Instead of just assuming static travel times according to the estimations under free-flow conditions, a dynamic approach is considered, calculating them according to the monitored saturation.

The travel time estimations for sections are calculated according to a formula from the Bureau of Public Roads[1] (Equation 7.1). The estimated mean travel time t_s for a section s is computed in dependence of the current traffic flow M as

$$t_s = t_f * (1 + a * (M/C)^b) \tag{7.1}$$

where $t_f = \frac{s}{v}$ denotes the travel time during free flow conditions based on the length s and the speed limit v of the section, and a and b are coefficients. Figure 7.2 depicts curves for several combinations of traffic flow and values for a and b with $C = 800$, $s = 1000$, and $v = 50$. Finally, C is the estimated maximal capacity of the section in terms of the number of vehicles, calculated according to formulas given by the U.S. Department of Transportation[2].

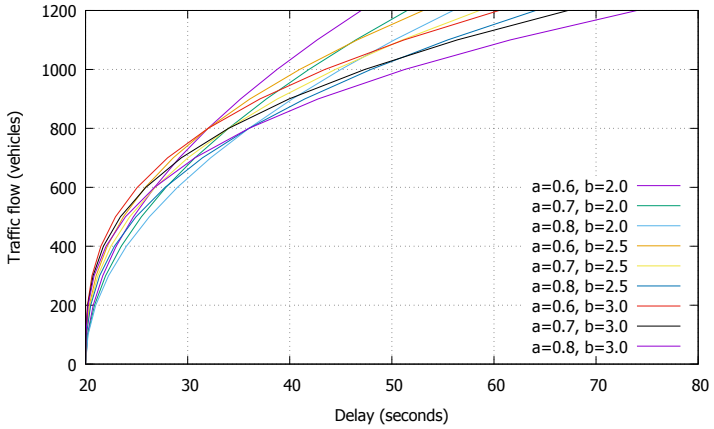


Figure 7.2: Estimated section travel time with the BPR formula for different traffic flow and a and b values.

The turning delays are approximated with a formula from [Web59] (Equation 7.2). Assuming that M corresponds to the turning's current traffic flow in vehicles per hour, S denoting the saturation flow (the maximal flow assuming permanent green), t_c representing the cycle time

[1] https://www.fhwa.dot.gov/planning/tmip/publications/other_reports/delay_volume_relations/ch04.cfm
[2] http://www.fhwa.dot.gov/ohim/hpmsmanl/appn7.cfm

of the intersection (in seconds), and t_g denoting the turning's effective green time (in seconds), the turning's delay t_d is calculated as

$$t_d = 0.9 * [\frac{t_c * (1 - t_g/t_c)^2}{2 * (1 - M/S)} + \frac{1800 * g^2}{M * (1 - g)}]. \tag{7.2}$$

Finally, $g = \frac{M}{t_g/t_c * S}$ corresponds to the degree of saturation of the turning for the current green time t_g and traffic flow M.

The result is an up-to-date description of the network's traffic state, from which routes to arbitrary destinations can be derived. Each TLC locally determines the routes with the lowest travel time which are then visualised through VMS's at each intersection. For now, only the route with the lowest travel time is displayed. However, the output can easily be extended to show alternative route recommendations.

7.4 Decentralised and adaptive routing protocols for urban road networks

Standard network protocols from the Internet domain, such as the distance vector routing (DVR) and the link state routing (LSR) protocol [Tan03] were adapted to road traffic guidance in previous works [PTL+12] and applied within the route guidance mechanism in OTC. Since these protocols work well for a complex network with a huge number of nodes, such as the Internet, it is deemed as an appropriate approach for urban road networks. In the following, the basic functionality of these reactive protocols are explained. The contribution in form of two anticipatory routing protocols based on traffic flow forecasts is presented.

7.4.1 Link state routing (LSR)

A modified version of the Internet protocol LSR serves as route guidance heuristic. Each TLC broadcasts the estimated travel times for each outgoing section and each of its local turning movements to other nearby controllers. Afterwards, each TLC constructs a network graph representing the topology of the road network and the estimated traffic conditions. Finally, the best routes through the network are derived. The protocol has a five-step process:

1) Find adjacent neighbours: Initially, an RC has no knowledge about the adjacent controllers he can communicate with. Therefore, an initial shake-hands message is sent to adjacent controllers. This step has to be executed only once, during the start-up phase.

2) Local delay estimation: Next, each RC periodically estimates the local delays for each turning of the local intersection. These delays are calculated based on the current waiting times during red light phases and the estimated travel time to a next intersection based on the current traffic conditions. Afterwards, these estimations are communicated to all other RCs in the network using broadcast messages (so-called *advertisements*). These link state advertisements

(LSA) contain *link states* describing a path from a starting intersection to a destination and its estimated travel time (see figure 7.3). Each LSA is assigned a sequence number, indicating if it is newer or older than another LSA sent by the sending RC. The message is only utilised in case it contains new information.

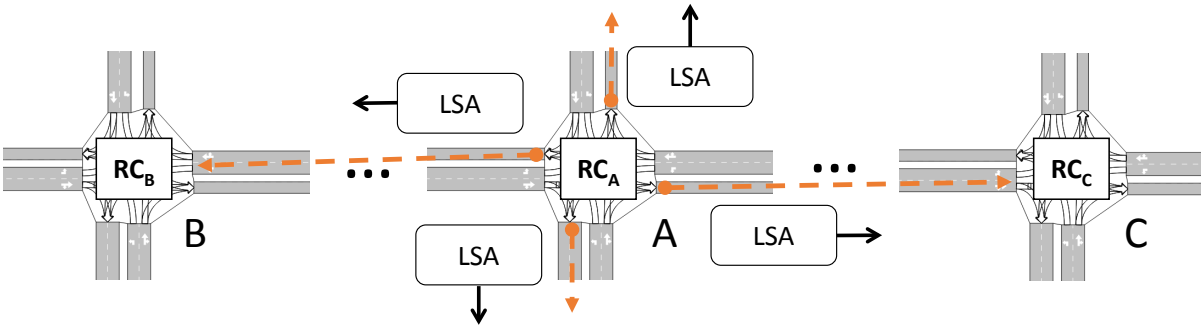


Figure 7.3: Routing component RC_A estimates the local delays for turnings and outgoing sections of intersection A. The resulting estimations are then sent to neighbouring RCs via link state advertisements.

3) Network graph creation: After receiving a full set of advertisements from all other RCs, each RC builds a graph by connecting the subgraphs obtained from the link states. This graph represents the topology, the current traffic flow, and the approximated travel times within the network.

4) Shortest paths computation: Every RC locally computes the best (for example the shortest or the fastest) routes from itself to other points of interest. The Dijkstra algorithm [Dij59] is used to calculate the paths with the lowest travel time from each RC to all reachable destinations based on the previously generated network graph.

5) Updating the routing tables: In a final step, each RC updates its interior routing tables. For each approaching road, one entry with the best route to all its reachable destinations is stored. Finally, each table entry contains information for each incoming section, the destination, the recommended next turning, and the estimated travel time to this destination.

Further details on the existing LSR mechanism for urban road networks are given in [PTL+12].

7.4.2 Temporal link state routing (TLSR)

The novel TLSR protocol resembles an extension of the basic LSR protocol, utilising both current traffic demands and forecasts of future traffic conditions. By broadcasting graph-series that encode current and forecasted traffic flow, TLSR is able to consider the time-dependant developments of traffic. The previously static network graph is converted into a time-dependent representation of the current and the future traffic conditions. Thus, the following adjustments are applied to the second, third, and fourth step of the previously introduced LSR protocol.

Local delay estimation and forecasting
The local delays are estimated both for the current traffic flow and for a number of traffic flow forecasts for future points in time (i.e. ten instances, one for every minute). The forecasts are created by the forecast module, as presented in section 2.4 (Figure 7.4). The number of forecasts, their interval, and the forecast horizon depends on the size of the road network. The more forecasts we create, the larger the packages that are sent over the network get. This leads to high latencies and message overhead. A forecast interval that is too small is not advisable since not enough sensor data will be available to correctly approximate the actual traffic state. The forecast horizon only has to span several minutes (up to 15 minutes) since the forecasts become less accurate the longer the horizon gets, due to the dynamic nature of urban traffic.

Based on previous forecasts and their respective actual values, error measures can be used to estimate the forecast accuracies and the standard deviations of the forecast errors. The forecasts and the forecast errors are added to the LSAs and are broadcasted additionally to other RCs.

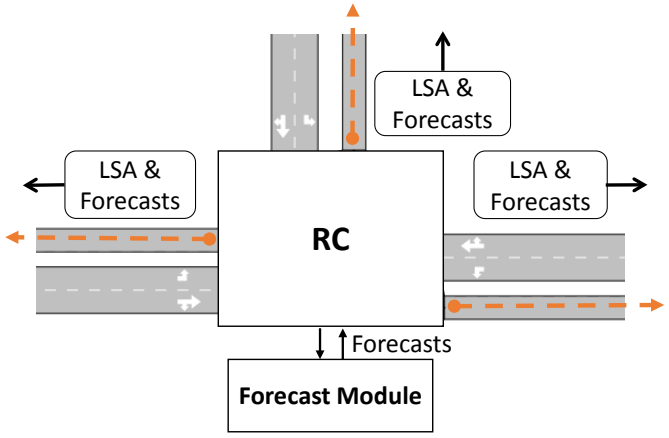


Figure 7.4: The routing component estimates the local delays for turnings and outgoing sections of intersection. Forecasts for future points in time are derived using the forecast module. The resulting estimations are sent to neighbouring RCs via link state advertisements.

Network graph creation
The edges (representing roads) of the network graph are extended. They contain both the current traffic flow and the forecasts for several points of time in the future in a fixed interval, defined by the interval in which the time series data is aggregated. Thereby, the resulting graph is converted into a *time-dependant representation* of the network.

Shortest paths computation
The benefit of including forecasts received from other RCs is highly dependant on their accuracy. Another important aspect is the degree to which they are taken into account for the calculation of the estimated travel times for the route recommendations. Thus, the Dijkstra algorithm is adapted to compute the routes with respect to the estimated precision of the forecasts.

Dijkstra algorithm considering forecast accuracy

The derivation of qualitative route recommendations depends heavily on the accuracy of the forecasts and the degree to which forecasts are taken into account. This ranges from only re-

lying on current traffic conditions (which is similar to the basic protocol) to considering long periods of forecasted traffic demands. To estimate the accuracy of the forecast F_t, the sending RC calculates the mean absolute scaled error (MASE) [HK06] (see section 6.3.3). Remember, a scaled error of less then one arises if the forecast is better than the average naive one-step forecast computed in-sample. The calculated result of the error measure is sent to other controllers which use it to determine the degree to which they consider the forecast, respectively the flow value. The lower the MASE, the higher the trust in the accuracy. Likewise, the influence of the forecast on the estimated travel time calculation increases. Finally, the current flow Y_t and the forecast F_{t+k} for time step $t + k$ are combined based on a smoothing function computed as

$$x_t^* = \alpha Y_t + (1 - \alpha) F_{t+k} \quad \text{where} \quad 0 \leq \alpha \tag{7.3}$$

with α being the MASE. In extreme cases, we only consider the forecasts or only the monitored sensor values. The first case occurs when the MASE value equals zero. The second case occurs when MASE has a value of one or higher. For this formula, MASE values above one are set to an upper limit of one. The combined result x_t^* then serves as estimated delay for the according turning or section.

Time-dependent Dijkstra algorithm considering forecasts

Previously, Dijkstra considered only one value per edge of the network. The edges are extended, containing both the estimated costs for the current time step and for several forecasts for different points in time. The modified Dijkstra algorithm chooses the closest entry for the point in time where a value is needed. A demonstrative example is shown in figure 7.5. The task is to estimate the travel time from intersection A to C based on the current travel time estimations and forecasts for future points in time. This means, at intersection A (start of the route), the current travel costs $t_A = t_0^{AB}$ at time $t = 0$ from intersection A to B are considered. At the next intersection B, the forecast $t_B = t_{t_A}^{BC}$ for the estimated arrival time $t = t_A$ from intersection B to C is used, and so on. The turnings delays are chosen accordingly. Finally, the approximated travel time for the whole route (here from A to C) based on the monitored sensor data combined with the forecasts is computed.

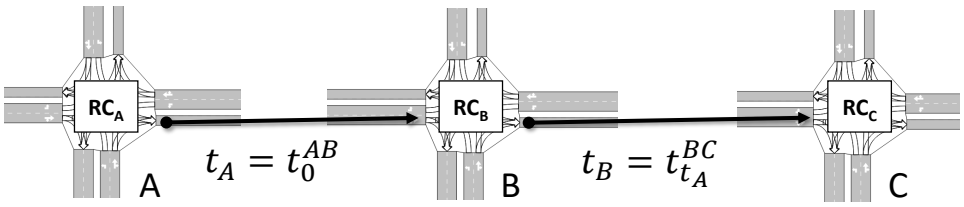


Figure 7.5: Estimation of the travel time from intersection A to C with time-dependant use of actual travel times and travel time forecasts.

7.4.3 Distance vector routing (DVR)

As an alternative, the DVR protocol has been considered. The DVR protocol maintains routing tables for each of the intersection's approaching sections containing the estimated travel times

to prominent destinations within the road network. RCs update their table entries based on messages received from neighbouring RCs, periodically communicating updates of the estimated travel time to its neighbours. This process works in an asynchronous and distributed fashion. The DVR standard protocol works as follows:

1) Find adjacent neighbours: An initial shake-hands protocol is initiated in order to find adjacent RCs. This step has to be executed only once during the start-up phase. The following steps are executed repeatedly in every iteration.

2) Create local routing tables: Initially, each intersection checks if it is directly connected to prominent destinations (such as the main hall). In case such a prominent destination is detected, the RC creates a new routing table entry for each approaching section leading to that destination. Each table entry contains the destination, the approaching road, the proposed next turning, and the estimated travel time towards the destination. The travel time is calculated based on the delay caused by red lights plus the estimated travel time to the destination. The travel time can be estimated in a static way, derived from the length and the speed limit of the connecting road, or dynamically, based on the current traffic flow (see section 7.3.2).

3) Distribution of routing tables: Newly created or updated table entries (so-called distance vectors) are then sent upstream to adjacent RCs where matching routing table entries are updated iteratively.

4) Routing table update: When the RC receives a distance vector, it checks its internal routing table. If a destination is yet unknown, a new entry is created, otherwise the existing entry is updated. Its costs and the proposed next turning are updated if the costs for the received route are lower than the previously stored information. In this case the message is further distributed. At last, each RC knows the fastest route with the lowest estimated travel time to all prominent destinations that are reachable from itself.

7.4.4 Temporal distance vector routing (TDVR)

TDVR tries to cover the time-dependant developments of traffic by considering traffic flow forecasts for future time steps. Similar to the standard DVR process, RCs forward the updated request to further RCs in their proximity. The routing table entries are calculated based on the current traffic flow conditions and the respective traffic flow forecasts. The previously described DVR protocol processes the road network upstream, going from a destination towards connected intersections. This process is not applicable for TDVR as each RC has to know the estimated travel time from an initial RC to itself to determine the point in time for which he has to compute a traffic flow forecast.

Referring to figure 7.5, to determine the travel time from RC_0 to RC_2 with respect to forecasts, the following steps have to be considered. First, the travel time (see section 7.3.2) from RC_0 to RC_1 has to be calculated. Then, RC_0 sends the message to RC_1. RC_1 receives the request and makes forecasts of the turning's delays and of the travel times for sections going to neighbouring

RCs for the point in time the vehicle is estimated to arrive at RC_1. Finally, the route from RC_0 to RC_2 is computed. As an additional step, the discovered route and its estimated travel time is returned to RC_0. RC_0 receives this information and updates its routing tables.

7.4.5 Adaptation for regional routing

The DRG mechanism relies on the collaboration between RCs within the road network. The exchange of messages leads to one significant problem. Broadcasting local traffic informations to other RCs leads to high communicational overhead which increases quadratic with the number of RCs. In the following, m denotes the number of prominent destinations and n represents the number of intersections in the network. In the worst case, DVR has to send n messages per destination, resulting in a communication complexity of $\mathcal{O}(n * m)$. The LSR protocol broadcasts the locally created link states of an intersection with a single message. As each controller has to forward the data of each other RC at most once (a message is dropped in case the sequence number is lower than the last received one), the communication complexity is $\mathcal{O}(n^2)$.

To minimise this overhead, the DVR and the LSR protocol were extended by Lyda [Lyd10], based on the concept of the Border-Gateway protocol [Tan03], also known from the Internet domain. The concept of the distinction between intra- and inter-network routing, separating larger networks into smaller sub-parts, was transferred to regions of cities. RCs in close proximity to each other form a region and each RC belongs to exactly one region. If all its neighbouring intersections are located in the same region, the RC is called *interior*, otherwise *exterior*. Only exterior RCs are allowed to communicate messages with exterior RCs from other regions. Consequently, RCs have to propagate less messages and the complexity for the calculation of the shortest paths decreases. However, interior RCs only have a limited view consisting of their sub-network. Compared to the standard protocols, the routing process within regions stays the same.

In this work, we adapted the concept of regional routing to the two novel protocols, TLSR and TDVR. In the following evaluation, they are compared to the regional DVR and LSR protocols.

Figure 7.6 shows a Manhattan-style network separated into three distinct regions A, B and C with 9 intersections each and 9, respectively 10, centroids (a centroid models a source/sink of traffic). Dark dots highlight exterior RCs. Centroids are represented by light circles. Lines between intersections show the ability to communicate with each other. The information being displayed on a VMS can also easily be adapted for this hierarchical organisation. Route recommendations to destinations in other regions are presented in the form “to region A turn right, duration 15 minutes”. Destinations in the same regions are explicitly addressed as before.

7.5 Evaluation

A simulation study helps us compare our forecast-augmented routing protocols TDVR and TLSR with their standard variants. Furthermore, the protocols are evaluated by comparing OTC-controlled intersections with and without DRG against a reference run with fixed-time signalisation. The simulation-based evaluation addresses the following questions:

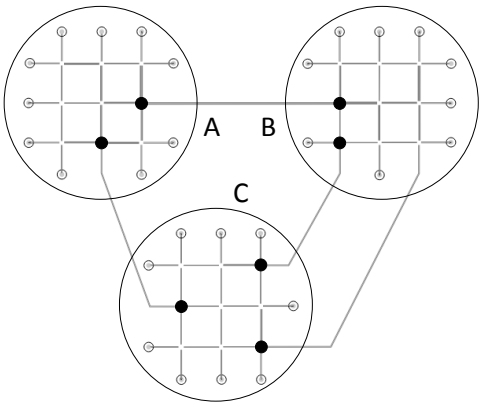


Figure 7.6: Road network with 3 connected regions. Dark circles highlight exterior nodes. Centroids are represented by light circles. All other intersections are interior nodes.

- Which strategy leads to the lowest travel times?
- Which strategy decreases emission the most?
- How do incidents and congestion influence the performance?
- How robust is the system under various compliance (driver acceptance) rates?
- How does the separation into regions influence the performance and the number of messages sent?

To simulate the traffic network, Aimsun 8.0 [BC02] is used, a professional traffic modelling and simulation software widely used by traffic experts. The simulated scenarios represent an artificial Manhattan-style road network (Figure 7.7) and a regional variant (Figure 7.12). Table 7.1 depicts the fixed-time signal plan deployed on each of the simulated traffic light controllers. The signal plan defines green times and interphases in between green phases. The time of yellow light is three seconds. The total cycle time of the signal plan is 90 seconds.

Table 7.1: Fixed-time signal plan for the evaluation scenarios. Signal timings are given in seconds (IP=Interphase, GT=Green time). The duration for yellow light is three seconds.

Phase 1		**Phase 2**		**Phase 3**		**Phase 4**		Cycle
IP	GT	IP	GT	IP	GT	IP	GT	time
10	7	23	5	10	7	23	5	90

The routing protocols are executed every two minutes. Each protocol is evaluated for different routing compliances (i.e. the degree to which drivers follow the route recommendations). As studies report varying rates [ESH07, ENR96], the approaches are evaluated for compliance rates of 10% (low), 40% (medium), and 70% (high). Cars not following the route recommendations of the DRG system, traverse the network according to the static shortest path to their destination without considering the traffic conditions. The proactive routing protocols create forecasts in intervals of 90 seconds for every section and turning.

7.5.1 Scenario I: A Manhattan-style road network

The first network consists of a 5-on-5 road network of 25 signalised intersections and 20 prominent destinations at the border of the network (Figure 7.7). In the following, the results for this network for an incident-free and a congested scenario are presented. As the data is gathered when the simulated vehicles have completed their trip, the effects of incidents show up in the figure with an approximate delay of ten minutes after the incident's occurrence.

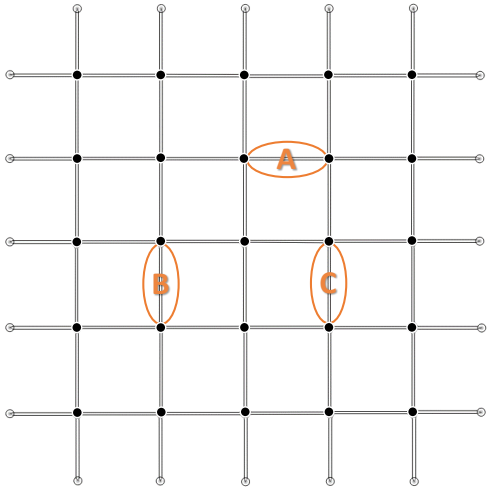


Figure 7.7: Manhattan-style road network with 25 signalised intersections (black circles) and 20 destinations (network-leaving sections). The ellipses mark the three incident locations (Incident A starts at 105, incident B at 45, and incident C at 75 minutes).

Incident-free scenario

The first scenario evaluates free-flowing conditions. Every hour, 15 vehicles travel from every origin to every destination, resulting in 5700 vehicles per hour traversing the network. The simulation duration was 2 hours and 15 minutes. Figure 7.8 indicates that OTC with and without routing is able to drastically reduce the average delay compared to fixed-time control (FTC). The figure shows the average travel time in seconds per kilometre for a simulated scenario of two hours. The compliance rate was set to 70%. The average travel time is given in brackets behind the name of the strategy.

The first 15 minutes of the simulation represent the warm-up time, where OTC gathers data to calibrate the forecast methods. Each observer/controller pair needs to populate its initial empty database of mappings between optimised signal plans and monitored traffic demands. Therefore, the performance during the warm-up time is identical for all approaches. Afterwards, we see that the static fixed-time signal plans do not reduce the negative impacts of the raising traffic demands.

Table 7.2 presents the average travel times over all trips, the average fuel consumption and the average CO_2 emissions per vehicle, comparing the reference run and the proactive routing protocols for different compliance rates. The fuel consumption is directly influenced by the length of the trip and the waiting times at red lights. The reduction compared to the reference

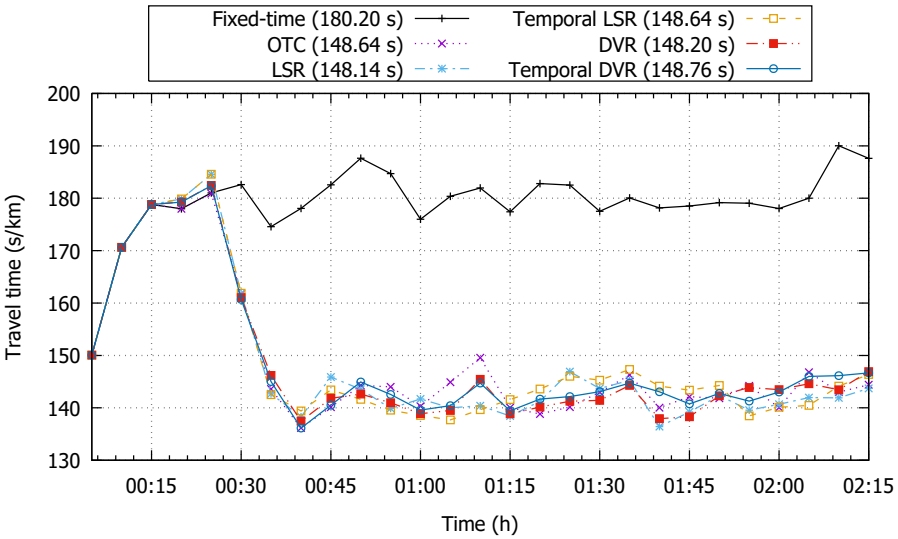


Figure 7.8: Travel times for the incident-free Manhattan scenario. The compliance rate for the routing protocols is 70%.

Table 7.2: Simulation results for the incident-free scenario with TDVR and TLSR. Values in brackets show the improvement compared to the fixed-time control (FTC).

	FTC	TDVR			TLSR		
		0.7	0.4	0.1	0.7	0.4	0.1
Travel time [s/veh]	280	228 (19%)	228 (19%)	228 (19%)	227 (19%)	230 (18%)	227 (19%)
Fuel [l/100km]	15.0	14.5 (3.3%)	14.4 (4.0%)	14.2 (5.3%)	14.5 (3.3%)	14.6 (2.7%)	14.2 (5.3%)
CO_2 [g/veh]	515	506 (1.7%)	505 (1.9%)	501 (2.7%)	505 (1.9%)	508 (1.4%)	502 (2.5%)

run is indicated in brackets. Not only is travel time reduced significantly (18%) in comparison to the reference run, but so are fuel consumption (3% to 5%) and emissions (2% to 3.3%). These results must be interpreted with caution. During undisturbed conditions, an improved signalisation alone is enough to guarantee a reduction of queues. OTC without routing already reduces the travel time to 225.3 seconds.

Congested scenario

To simulate disturbed traffic conditions, road blockages are investigated. The congested scenario simulates disturbed traffic flow through temporary blockages of roads, resulting in traffic congestion. In figure 7.7 the locations of these incidents are highlighted with circles. Three streets were blocked for 40 minutes each, forcing vehicles to take alternative routes. Incident 1 starts at 15 minutes, incident 2 at 45 minutes, and incident 3 at 75 minutes. Ten vehicles per hour travel from every origin to every destination which results in 3800 vehicles per hour, traversing the network.

In contrast to the undisturbed scenario, the congested scenario clearly shows the benefit of dynamic route guidance, especially during incidents. Table 7.3 shows the results. The routing

mechanism tells drivers how to avoid congested areas. They reduce average travel time by 10% (TLSR) to 11% (TDVR) (Figure 7.9), assuming a compliance rate of 70%.

Table 7.3: Simulation results for the congested scenario with TDVR and TLSR under different compliance rates.

	FTC	TDVR			TLSR		
		0.7	0.4	0.1	0.7	0.4	0.1
Travel time [s/veh]	345	306 (11.3%)	310 (10.1%)	320 (7.2%)	310 (10.1%)	313 (9.2%)	335 (2.9%)
Fuel [l/100km]	16.6	16.4 (1.2%)	16.3 (1.8%)	16.6 (0.0%)	16.6 (0.0%)	16.4 (1.2%)	17.0 (-2.4%)
CO_2 [g/veh]	549.6	543.6 (1.1%)	545.7 (0.7%)	551.8 (-0.4%)	547.2 (0.4%)	547.0 (0.5%)	562.3 (-2.3%)

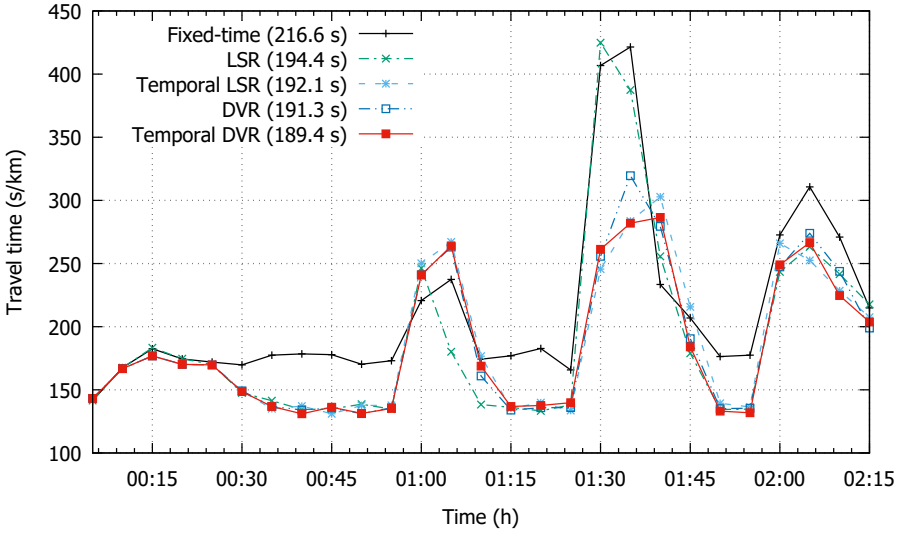


Figure 7.9: Travel times for the congested Manhattan scenario. Compliance rate for the routing protocols is 70%.

The best performance was delivered by TDVR, achieving the highest travel time reduction of all protocols. At 1:45, during a severe incident, TDVR reduces the average travel time to 282 seconds per kilometre (33% improvement over the reference run with 421 seconds and 30% improvement over OTC without routing with 408 seconds). This indicates that TDVR correctly forecasted the upcoming congestion due to the incident, preventing more severe disturbances. This resembles an improvement of 4.0% compared to OTC without routing, with an average trip travel time of 320 seconds. The decrease in travel time is achieved by re-routing drivers over alternative routes. These routes can be longer than the planned one and therefore, the use of routing protocols sometimes leads to slightly higher fuel consumption and CO_2 emissions. However, DRG promotes the reduction of greenhouse gas emissions by reducing the number of start-stops. The traffic network recovers faster towards an undisturbed state compared to the fixed-time control, making the road network more reliable. We call this characteristic *robustness*, meaning that the routing protocols are more robust against disturbances.

Figure 7.10 and figure 7.11 present the average travel times evaluated for several compliance rates. The figure's horizontal axis shows the simulation time and the vertical axis shows the average travel time in seconds per kilometre. A higher compliance rate means that drivers are

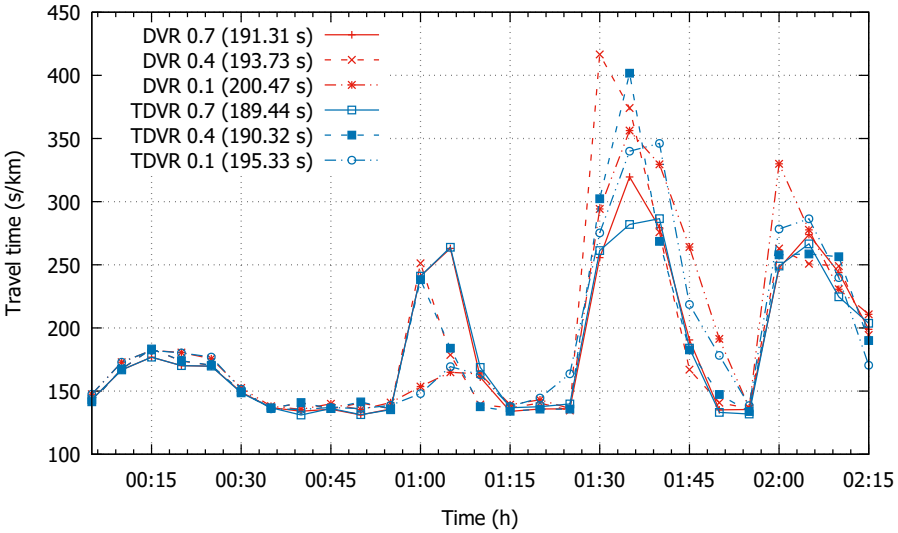


Figure 7.10: Travel times for the congested Manhattan scenario with DVR and different compliance rates. The average speed ranges lies between 18 and 20 km/h. Higher compliance rates reduce the average travel time.

more likely to follow the given routing proposals. The results suggest that the benefit of all routing protocols increases for higher compliance rates.

7.5.2 Scenario II: Regional Manhattan-style road network

The second network (Figure 7.12) allows for an evaluation of regional routing protocols. Three equally shaped 3-on-3 Manhattan-style regions with nine signalised intersections are each connected by one or two streets. The simulation time spanned two hours. For every origin-destination pair, eight vehicles per hour are generated, resulting in 6048 cars per hour traversing the network.

The traffic flow forecasts were created with the following methods: Exponential smoothing, double exponential smoothing, double smoothing average, moving average, and a kalman filter. Their forecasts were combined with the simple average method. The protocols have been evaluated with respect to the vehicles' mean delay and the mean travel time averaged over all finished trips. The fuel consumption and pollution emissions of the simulated vehicles have been investigated to estimate the environmental impact of DRG. The emissions have been determined with the help of AIMSUN's environmental model, which is configured according to [PBL06].

Table 7.4 depicts the comparison of communicational and computational effort between regional protocols and the basic variants. The table shows the number of messages each TLC has to send during one iteration of the executed routing protocol, as well as the average runtime of a complete protocol run in seconds. The results clearly show that the regional protocols decrease the number of messages as well as the computational overhead. The reference run with FTC has no communication between TLCs and therefore does not have to send any messages. The

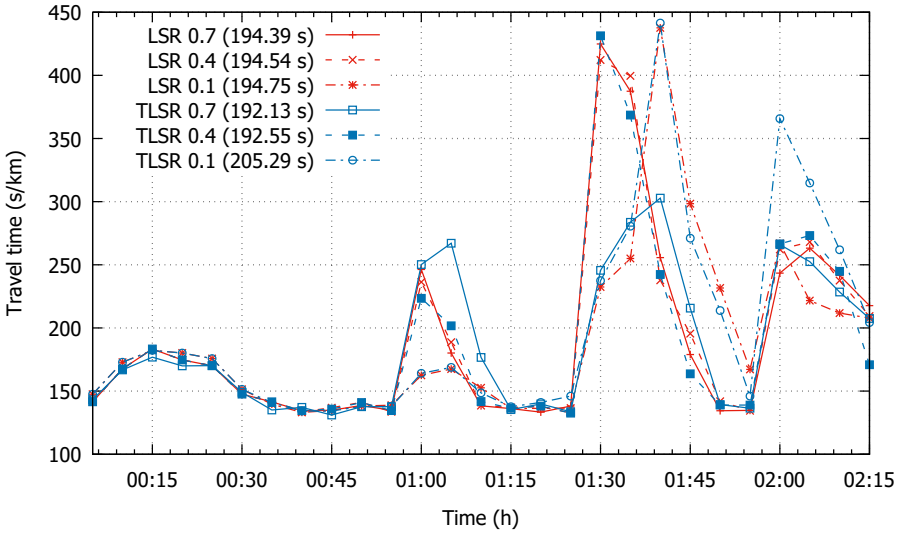


Figure 7.11: Travel times for the congested Manhattan scenario with LSR and different compliance rates. The average speed lies between 17 and 19 km/h. Higher compliance rates reduce the average travel time.

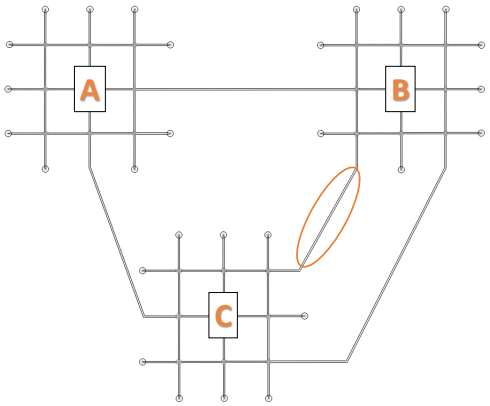


Figure 7.12: Manhattan-style road network with 3 connected regions. Dark dots mark exterior, light dots interior nodes. The ellipse highlights an incident created at minute 15, lasting for 20 minutes.

anticipatory protocols need more computational power to compute the forecasts for all sections and turnings of the network, leading to longer runtime. The message overhead can be reduced by increasing the interval between forecasts while reducing the number of forecasts. Finally, forecasts which represent similar values can be subsumed into a single value as they do not encode additional information.

At last, the regional and the standard protocols are compared for free-flowing and congested conditions. A congestion was created by blocking the section highlighted in figure 7.12 for 20 minutes. As table 7.5 indicates, the regional protocols do not, or only to a slight extend, increase the average travel time. The simplification of the communication due to the regional aggrega-

Table 7.4: Comparison between regional and standard protocols in terms of sent messages and computational time.

	FTC	DVR	Reg. DVR	TDVR	Reg. TDVR	LSR	Reg. LSR	TLSR	Reg. TLSR
Messages (#/RC/iter.)	0	1437	133	1463	133	35	25	35	24
Runtime (s)	10	360	105	1400	400	280	270	330	370

tion of RCs offers equally good route recommendations while reducing the communication and computational effort.

Table 7.5: Travel time of the regional and standard protocols for the regional scenario. The compliance rate for the routing protocols is 70%.

	DVR	Reg. DVR	TDVR	Reg. TDVR	LSR	Reg. LSR	TLSR	Reg. TLSR
Travel time (s/veh)								
Undisturbed	119.83	119.27	119.49	119.30	119.73	119.76	119.39	121.68
Disturbed	127.83	128.44	128.41	127.15	128.69	129.14	129.00	130.18

In summary, the evaluation results indicate that proactive route guidance lowers overall travel time and delays for road users. The algorithms are evaluated with several compliance rates, showing the benefit of our approach even under low compliance rates. In accordance to the findings of Fu [Fu01], our evaluation shows that reactive protocols and especially proactive approaches lower the average travel time in urban road networks.

7.6 Summary

Two novel time-dependent, anticipatory, and eco-friendly routing protocols for dynamic, proactive traffic guidance in urban road networks were presented: temporal link state routing (TLSR) and temporal distance vector routing (TDVR). The well-known Internet protocols DVR and LSR have been extended, utilising traffic flow forecasts to compute the best routes through the road network. The routes are determined by a self-organised approach, extending parametrisable traffic light controllers. The route recommendations are visualised by VMS's at each intersection, guiding drivers from intersection to intersection on a hop-to-hop basis.

A simulation study investigated the benefits of these protocols under disturbed and undisturbed conditions in two different networks with compliance rates of 10%, 40%, and 70%. Our findings strongly support the argument that traffic flow forecasts lead to a decrease in system-wide travel times for urban vehicular traffic. Consequently, this leads to a reduction in emissions and fuel consumption. In general, this counts especially for disturbed and congested conditions, but to a limited extend also for medium and low traffic saturations. The benefit increases for higher compliance rates. With undisturbed conditions, an improved signalisation alone is

enough to guarantee a reduction of congestion. The dynamic route guidance improves the network's robustness by guiding drivers on alternative routes avoiding blocked areas. The TDVR protocol showed to be the most beneficial approach, not only for congested but also for free flow conditions. The communicational overhead and the computational costs can be reduced by partitioning a larger network into smaller sub-networks using the border-gateway protocol.

Momentarily, the DRG mechanism estimates the current and future mean travel times for each section. This information can also be used to estimate the congestion likelihood for sections. A simple approach would be to define fixed thresholds, whereas mean travel times higher than this threshold are defined as congested conditions and lower values as free flowing traffic. A traffic engineer can define these thresholds based on historic data. Obviously, this approach is prone to ageing and requires a lot of expertise. It is not far-fetched to assume that this approach would result in a rather high false alarm rate or low sensitivity. Utilising a reliable congestion detection mechanism could be beneficial for the acceptance of the routes proposed by the DRG. By displaying additional congestion alarms on the VMSs, the compliance rates can be increased. The following chapter presents how machine learning concepts can be applied to the congestion detection problem.

Chapter 8

Automatic road traffic congestion detection

Current studies predict that more than 60% of the world population will live in cities in 2030. This will consequently attribute to a greater number of traffic-related problems. According to several reports [JvM09, SEL15, LNK+10], the number of kilometres driven and the delay due to congestion increased over the last decades and this trend is assumed to last. The urban mobility report of 2015 [SEL15] has documented that the yearly delay per commuter stuck in congestion sums up to 42 hours (38 hours in 2012). This results in a waste of 3 billion gallons of fuel in total and 380 lbs of CO_2 per commuter. Furthermore, road safety and crash frequency are affected by the occurrence of congestion [MW10]. Crash frequency seems to increase especially during unstable traffic conditions with high variability of density and speed, a condition which accounts for the establishment of congestion.

Especially in urban areas where space is limited, one has to rely on the existing infrastructure. Furthermore, the technology used for traffic management needs to be optimised for a better utilisation of the existing roads. Urban areas with a large number of road intersections are prone to blocking back, for example "congestion on a road link may cause a tailback that blocks other routes that are not overloaded" [ISY04]. ATCSs with incident management components are one approach for the former problem. This includes the collection of sensor data, the detection of present and the prediction of upcoming congestion, as well as the congestion management in terms of actions to take.

It has to be noted that the identification of congestion is more feasible on highways, as the shock waves theory [HSA10] can be applied. In contrast, in urban areas, the relationship between different traffic streams is more complex. Incidents can occur due to accidents, lane closures, or long-lasting road works. Furthermore, two third of traffic delays are caused by spontaneous, unexpected, non-recurring incidents.

Still, there are no consistent definitions or measures for congestion, especially not in urban areas [Ber05]. Traffic quality is often classified into one of six levels of service which are denoted by the letters A (free flowing traffic) to F (congested conditions). Congestion is sometimes also defined as stop-start conditions, a complete stop for at least five minutes, or travelling at less than the speed limit [Ber05].

The difficulty of automatic congestion detection for machine learning lies in the fact that non-congested situations have a much higher possibility than congestion events. Consequently, it is difficult for any algorithm to learn what congestion looks like as the training sets usually only have a very small number of positive examples. Furthermore, future congestion may look totally different compared to the learned representations.

In the following, an introduction to incident management as executed by real-world ATCSs is given. Afterwards, the state-of-the-art in incident detection algorithms is summarised. I will apply several machine learning algorithms to the automated incident detection on motorways and arterial roads. Finally, an evaluation with real-world sensor data demonstrates the performance of the proposed methods.

8.1 Incident management in real-world traffic management systems

Incident detection is usually executed with automatic incident detection (AID) algorithms or by manual evaluation. Its objectives are to provide warnings and information about the occurrence of incidents for traffic participants. The term *incident detection* is defined as "the process of identifying the spatial and temporal coordinates of an incident" [OK99]. It has to be noted that most incident detection algorithms (despite their name) do not detect incidents itself but congestion which can be caused by an incident or by a recurrent temporary bottleneck. These types of breakdowns are typically occurring at bottleneck locations in the traffic network. Usually, breakdowns last throughout the peak period if traffic volumes are close to or above road capacity. The 2010 federal highway administration traffic incident management handbook [OAS+10] defines an *incident* as "any non-recurring event that causes a reduction of roadway capacity or an abnormal increase in demand. Such events include traffic crashes, disabled vehicles, spilled cargo, highway maintenance and reconstruction projects, and special non-emergency events". Incident detection is just one part of the incident management process. Figure 8.1 depicts the complete standard process flow [DCG12].

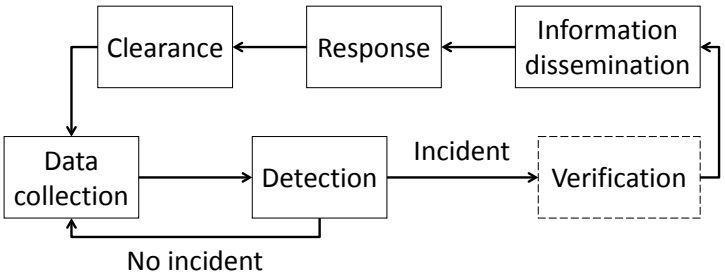


Figure 8.1: Typical flow chart of the AID process starting with data collection. The verification step is optional.

First, data sources, such as CCTV cameras, floating cars, or inductive loop detectors monitor the traffic flow, and provide a situation description of the current traffic condition. Second, this information is usually sent to a control centre where the processing of this data is taking place. Third, a traffic analysis is executed by incident detection algorithms. A useful description of an incident should contain its type, its exact location, its severity, and the time of occurrence. Fourth, an optional verification of the incident alarm can take place. Fifth, this real-time traveller information has to be disseminated among the traffic participants. The congestion management can be done manually by traffic experts through adaptation of signal plans or by providing additional information (alternative routes and estimated travel times) via VMS, radio broadcasts, and internet services. Finally, clearance procedures have to be initiated to restore the conditions before the incident.

In the following, a brief overview of incident management in traffic management systems is given. They are installed in reality and able to offer some reaction to oversaturated conditions.

SCOOT

One example for a real-world traffic control system including incident management is SCOOT [RB91]. SCOOT automatically identifies critical links causing congestion, and reacts with a recommendation of strategies [BUW+01]. SCOOT assumes the presence of congestion when the detector, related to an upstream intersection of the respective road, monitors a stationary queue. It will respond autonomously by extending the green time to the congested road, or by increasing the cycle time at the nearby signalised intersection. The maximal length of additional green time is defined during setup of the system by assigning a congestion importance factor to each signalised intersection. SCOOT thereby reduces the queue, preventing further blocking of exiting streets. However, the real-world implementation usually requires additional field-studies and careful parameter adjustment.

COMPASS

COMPASS [MW91] is another advanced traffic management system. It relies on sensor technology to monitor the traffic conditions, software to analyse the traffic conditions, and plans defining which actions to take. All information is gathered in a central traffic operation centre. The incident detection is executed by the *all purpose incident detection* (APID) algorithm. APID extends the california 7 algorithm, an incident detection algorithm based on a binary decision tree. APID executes separate routines for light, medium, and heavy traffic, as well as persistence tests to reduce false alarms. Further management actions during incident situations have to be executed by humans. However, dynamic message signs are automatically displaying up-to-date information about the level of congestion.

OPAC

The Optimized Policies for Adaptive Control (OPAC) system [GPA02] offers two distinct modes to react to oversaturated conditions. For an isolated signalised intersection, it extends the green time for phases exceeding a user-specified threshold. The coordinated mode respects the requirements of several intersections. Besides extending green times, the cycle times can also be adjusted.

SCATS

SCATS [SD80] is equipped with a centralised unusual congestion server which receives updates of the monitored traffic data in real time. It generates alerts in case a road is classified as congested by its monitoring tool. Hereby, traffic volumes and the degree of saturation monitored by loop detectors are taken into account. Again, countermeasures have to be taken manually by traffic experts [SS10].

8.2 Algorithms for incident detection

Comprehensive reviews of incident detection algorithms, their deployment, and detector technology are given by [PX06] and [MHZP99]. They divide incident detection algorithms into three categories: *roadway-based*, *probe-based*, and *driver-based*. Roadway-based algorithms use traffic flow data measured at certain points along the road based on detector technologies, such as inductive loop detectors or infra-red technology. Probe-based methods use data from vehicles equipped with special transponders and receivers. Driver-based algorithms rely on information about incidents given by people, without using computational devices. However, incident detection algorithms can also be classified into *point-based* and *spatial measurement-based* algorithms [OK99]. The family of point-based algorithms can be further separated into *comparative algorithms*, *statistical processing*, *traffic modelling and theoretical algorithms*, and *advanced machine learning algorithms*. Spatial measurement-based algorithms make use of CCTV cameras and image processing algorithms. Point-based algorithms are usually deployed on motorways [YSS04], whereas spatial measurement-based technology is also used in urban traffic networks [ZX10].

8.2.1 Point-based incident detection algorithms

Comparative algorithms: Algorithms of this class compare the current traffic conditions to predefined, static thresholds. They use traffic parameters, such as occupancy, speed, or volume, to classify the current traffic conditions with the help of binary decision trees. An incident alarm is raised in case the monitored values exceed certain thresholds. Representatives are the algorithms from the class of California algorithms [PK76], and the all purpose incident detection (APID) algorithm [MW91].

Statistical processing: Statistical algorithms use standard statistical techniques to detect whether the model obtained from monitored traffic flow patterns differs from the estimated traffic characteristics. The standard normal deviate algorithm [DMN74] and the Bayesian algorithm [LK78] try to detect sudden change in traffic flow by comparing historical occupancy values to the current traffic conditions. A sudden change is seen as an indicator for the occurrence of an incident.

Traffic modelling and theoretical algorithms: Traffic modelling and theoretical algorithms apply the basics of traffic theory to describe and detect incident conditions. The fundamental relationships between several traffic parameters are exploited to discriminate between incident-free situations and incidents. A well-known algorithm of this class is the McMaster algorithm which makes use of the catastrophe theory model [GH89]. Based on the current volume-occupancy ratio derived from detector stations, the third version of the algorithm uses three distinct categories (uncongested, incident, congested) to classify the monitored traffic conditions.

Advanced machine learning algorithms: Machine learning techniques use artificial intelligence to learn patterns or functions from a training set of data [HTF01]. A number of authors applied different machine learning algorithms to the problem of incident detection, such as support vector machines [DKKT14], artificial neural networks [SJC04], and fuzzy logic algorithms [Bru10]. In terms of calibration, they usually need less efforts than threshold-based algorithms which often require an excessive amount of calibration as the algorithm thresholds can vary across detector stations. However, their advantage is often bought with an increase in computational costs.

8.2.2 Spatial measurement-based incident detection algorithms

Aerial sensing imagery from satellites and video streams from CCTV cameras provide image data that is automatically analysed by video-image processing algorithms. These still frames and video streams allow for the detection of incidents caused by single vehicles and the extraction of traffic data, such as flow, speed, and occupancy. The AUTOSCOPE system [MJ92] analyses video streams provided by cameras along motorways. It allows for vehicle detection and traffic parameter extraction. The incident detection is executed by the AUTOSCOPE incident detection algorithm (AIDA) [Mic91]. In case an incident alarm is raised, the information is provided to an incident management operator which has to determine the appropriate response.

8.2.3 Model fusion

Instead of just relying on one single incident detection technique, some researchers combine multiple methods (heterogeneous ensemble) or several instances of the same model with different parametrisations or inputs (homogeneous ensemble). These ensembles make use of methods, such as artificial neural networks [CWQL07], multiple naïve Bayes classifiers [LLCZ14], or support vector machines [ZGQ08]. The combination of several methods appears to result in higher detection rates and lower false alarm rates.

8.2.4 Incident forecasting algorithms

Only few papers on the topic of incident and congestion forecasting can be found, especially before 2000. In recent years, research in this field is slightly more vivid.

Kurihara et al. [KTN+09] propose a technique which relies on modelling the behaviour of ants and their use of pheromones for communication and route optimisation. Controllers located at intersections (so-called road agents) receive monitored traffic flow from nearby sensors to estimate the traffic flow density and to make short-term forecasting of traffic congestion at one minute intervals. An evaluation with a simulation of a manhattan-style road network showed that this technique can offer higher accuracy in congestion forecasts than the conventional statistical approach used by *Balaji et al.* [BSST07]. *Hiri-o-Tappa et al.* [HOTNPNPA07] use dynamic time warping to calculate an indicator for the congestion likelihood comparing speed data measured by loop detectors with historic data. They admit that their approach lacks in terms of meantime to detect and false alarm rate. Other researchers propose distributed congestion detection mechanism with vehicles equipped with wireless communication hardware

[HSA10]. This setup allows to detect shock waves, which propagate the building of congestion, by measuring the velocity of the surrounding traffic and the distance between following and leading vehicles. Others apply machine learning models to the congestion prediction problem [Hui06, LDW+09]. Beyond others, these methods include multi-linear regression models, ARIMA models, artificial neural networks, and radial basis functions. *Huisken* [Hui06] states that self-organizing maps, fuzzy logic, and ARMA models are not suitable for this problem, whereas supervised ANNs offer the best results.

8.3 Congestion detection as machine learning problem

> "Stuck in traffic is not an excuse. It's a sign of bad planning."
>
> Susan Elizabeth Phillips

In the following, we focus on the detection of congestion in contrast to detection of incidents. Congestion detection is a complex problem and poses several challenges. First, traffic congestion is a dynamic, location-dependent, non-linear phenomenon [Hui06]. Second, congestion detection is a classic example of imbalanced or skewed data in real-world applications [HG09, ZCWL13]. The imbalances are intrinsic, as it can be assumed that free-flowing traffic conditions are likely to have much higher probability than congested conditions. Third, the data monitored by detectors is often faulty and unreliable.

The inference of patterns from data, here the presence or absence of congestion, is a typical binary classification task. In this case traffic is either seen as congested or free flowing. Typically, the class distribution is not uniform among these classes. The dominating majority (negative) class is represented by the data points representing free-flowing traffic. The minority (positive) class is represented by rare instances of data points showing congested conditions. This imbalance and the resulting lack of training instances makes the learning process more difficult (all data sets are sparse in high-dimensional search spaces). For a two-class classification problem, the task can be expressed more formally as

$$f(\vec{x}) \rightarrow c_i \in \{i = 0, 1\}. \tag{8.1}$$

A feature vector $\vec{x} = \{x_1, x_2, \ldots, x_n\}$ is processed by a function f which maps the input variables to a specific class c_i (the label). In case of congestion detection, $\vec{x}$ contains a defined set of traffic parameters (such as average speed, detector occupancy, or number of passing vehicles within a certain time interval). Algorithms for pattern recognition are usually trained from labelled data where the individual observations are correctly classified (supervised learning) [JWHT13]. The goal is to fit a model that relates the observations in $\vec{x}$ to the correct class label c_i. In contrast, unsupervised learning tries to discover previously unknown patterns from unlabelled data.

Machine learning algorithms [Alp08] try to deduce a process, model, or function from observations to describe a certain behaviour. Some of these algorithms are able to learn new patterns and to improve their model at runtime (reinforcement learning), such as learning classifier systems [BBMBK08]. These algorithms choose and execute actions in reaction to the observations

and adjust their parameters, and their internal model according to the received feedback. In the following, the traffic congestion detection problem is treated as a task of learning classifiers from imbalanced data. Consequently, the optimisation objective is to learn a model that approximates the underlying classification task optimally.

8.4 Congestion detection with the XCSR

In the following, the XCS for real-valued inputs (XCSR) [Wil00] is applied to the automatic detection of congestion. Until now, the distributed, adaptive control of signalisation is the only adaptation of an LCS derivative within the traffic domain [BST+04, PRT+08]. In this section, the focus lies on the necessary adaptations in contrast to the standard XCSR as explained in section 4.1. XCSR's task is to learn which sensor values resemble congested conditions and to classify the given measurements into congested and uncongested (Figure 8.2). The underlying task is mapped to a single-step problem, as the according reward (the actual state of traffic) for action a_i is available in the next step. Following the standard configuration, the reward is either 1000 for a correct or 0 for a false classification. The action resembles the according predicted congestion classification (0 for congested and 1 for free-flowing).

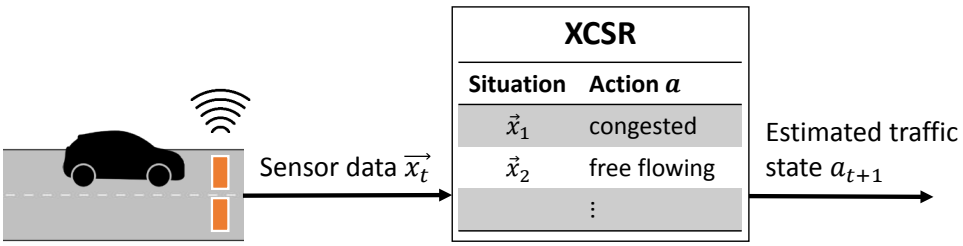


Figure 8.2: Congestion detection with the XCSR. For simplification, only the rule base of XCSR is shown.

8.4.1 Data collection

In every time step t, the current measurements from traffic detectors are retrieved. Thus, our approach falls into the class of roadway-based, respectively point-based algorithms. In the context of AID, this situation description is a vector $\vec{x} = (x_1, \ldots, x_n)$ of continuous, real-valued traffic parameters monitored by a sensor, such as a loop detector or a CCTV camera. $\vec{x_t}$ represents the current traffic condition at the local intersection or section. For the experiments, the sensor input is simulated by reading the next line from a data set. In case not every available sensor value is needed, it can be defined which ones to include and which ones to be omitted (for example we could only be interested in the average speed and occupancy). Finally, all remaining components of the resulting feature vector $\vec{f_t}$ have to be normalised to the range $0.0 \leq x_i < 1.0$ (XCSR demands the input values to be in this range). The normalised input vector is then forwarded to the XCSR for classification.

8.4.2 Training and rule discovery

XCSR is an on-line learning algorithm. For a valid comparison with other algorithms, a training phase is simulated before testing. During training, XCSR *explores* the situation space. Afterwards, the gained knowledge is *exploited*. The training is supervised, thus, each situation vector of the training set has to be classified into one of two classes, congested or free flowing. The training examples are given in the form $\{(x_i, y_i)\}$ such that x_i is the feature vector and y_i its class label. Amongst other factors, the exploration rate strongly depends on the chosen action-selection regime, and the execution rate of the GA. Usually, the action-selection method during exploration is random. For faster convergence, a fitness-proportionate selection is utilised, also known as roulette-wheel selection. The GA is triggered every few steps, and as a result, a new classifier is added to the population. After the execution of the chosen action, the reward is returned and reinforcement takes place.

8.4.3 Testing and evaluation

After completing the training phase, XCSR fully relies on the previously learned knowledge. During testing, the GA is no longer executed and the action-selection regime is switched to a deterministic best-action selection. The best action is the one with the highest fitness-weighted score in $[P]$. In case of missing actions in $[M]$, covering is executed. Thereby, new classifiers can still be created and added to the population.

8.5 Support vector machines for congestion detection

Support vector machines (SVM) [Alp08] are binary classifiers which provide good generalisation and convergence for classification tasks, and are able to handle high-dimensional data. A SVM is learned with a training set $\{(x_1, y_1), \ldots, (x_n, y_n))\}$ where x_i are the feature vectors and $y_i \in \{-1; +1\}$ are the binary classifications or labels of those instances. The goal of this training process is to build an optimal hyperplane, a linear discriminant that separates the sample's classes by the largest margin (Figure 8.3). Finding this maximum margin can be transferred to solving the quadratic programming problem which is mostly solved using sequential minimal optimisation.

Their performance is dependent on a coefficient C defining the margin between the two distinct classes and the kernel hyper-parameter γ. A large C gives a low bias and high variance because the cost of misclassification is penalised more. In contrast, a small C returns in higher bias and lower variance. The parameter γ of the Gaussian kernel handles the non-linear classification. A small γ gives a low bias and high variance, while a large γ results in higher bias and low variance.

In terms of congestion detection, the feature vector consists of locally available attributes, describing the traffic flow at the particular intersection or section. Usually, these attributes consist of the recent speed, volume, and occupancy values [PX06, DKKT14]. Given this feature vector, the SVM decides if the current traffic condition is congestion-free or congestion-bound, depending on which side of the hyperplane the feature vector is located.

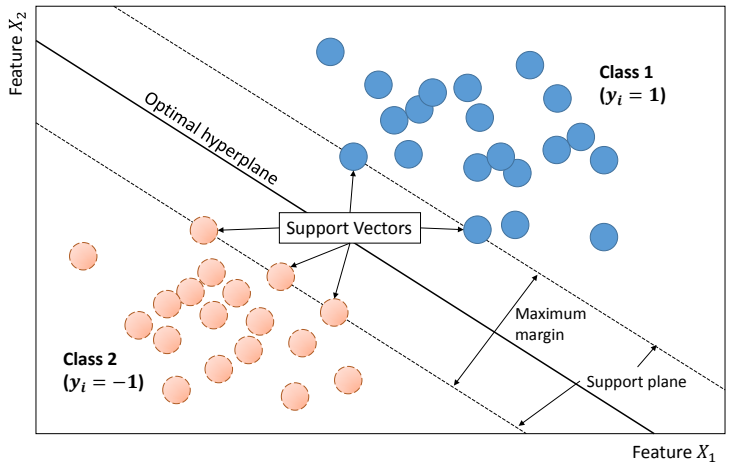


Figure 8.3: Schematic illustration of a support vector machine.

The following SVM variants were chosen for the evaluation: LaSVM, LaSVM-I, and SDCA. Their implementations are readily available from the *JKernelMachines* framework [PTC13]. *LaSVM* [BEWB05] is an efficient SVM solver that uses on-line approximation. It is able to handle noisy data sets using less memory than other state-of-the-art SVM solvers. *LaSVM-I* [EBG11] is an optimisation of LaSVM. It filters outliers based on approximating non-convex behaviour in convex optimisation. LaSVM-I proves to be faster in terms of training times, needing fewer support vectors, and offering only slightly worse accuracy. This is especially profitable during on-line learning. Stochastic dual coordinate ascent (*SDCA*) [SSZ13] is a method for solving large-scale supervised learning problems formulated as minimisation of the convex loss functions. It executes iterative, random coordinate updates to maximize the dual objective.

8.6 Evaluation

In the following, the experimental setup is described and the results of the conducted experiments are presented.

8.6.1 Experimental setup: Traffic data

The evaluation is done with real-world data sets provided by the Minnesota Department of Transportation (MnDOT)[1]. The data was recorded by inductive loop detectors in the vicinity of Minneapolis, averaged over five minute intervals, resulting in 2016 data points per week. Each data point contains the time of recording, average speed, volume, occupancy, and density. The data was manually classified as congested or not congested. A data point is assumed congested when the average speed drops below 35 mph and the occupancy rises at the same time. Each data set is split into three weeks for training (6048 data points) and one week for testing (2016 data points). This experimental setup was chosen over a simulation-based evaluation in OTC, as this allows to use realistic data over a longer time span.

[1] http://www.dot.state.mn.us/tmc/trafficinfo/developers.html

Figure 8.4: Map of Minneapolis showing the locations of the three detector stations (marked with red circles).

The locations and dates of monitoring are: Interstate I35E (Figure 8.5), June 2013 (detectors 2442, 2443, 2444, 2447, 2448); Interstate TH5 (Figure 8.7), December 2015 (detector 1577); and Interstate I94 (Figure 8.6), May 2015 (detectors 569, 365, 366, 367). The locations and dates are selected randomly. Traffic on I35E and I94 shows some typical seasonal behaviour. Congestion usually occurs during rush hour in the morning and evening on work days. In contrast, traffic on TH5 exhibits stop-and-go behaviour in the early hours during work days. MnDOT defines congestion as traffic flowing at speeds below 45 miles per hour. The selected data sets exhibit congested conditions between 2% and 29% of the time.

Figure 8.5: Aerial photo of Interstate I35E showing the locations of the selected detector stations. (© Google Maps, 2016)

Figure 8.8 shows a representative traffic flow profile on I35E, measured by detector station 2447 on Wednesday, 2013-06-12. The recommended speed is 60 mph (dotted line). The plot depicts a typical weekday morning rush hour from 7.30 a.m. to 9.30 a.m. The detector records faulty values from 7 p.m. to midnight, an issue to be faced when using real-world data from inductive loop detectors [PX06]. Faulty measurements are not removed, as it is interesting to see if the selected algorithms are able to deal with this problem.

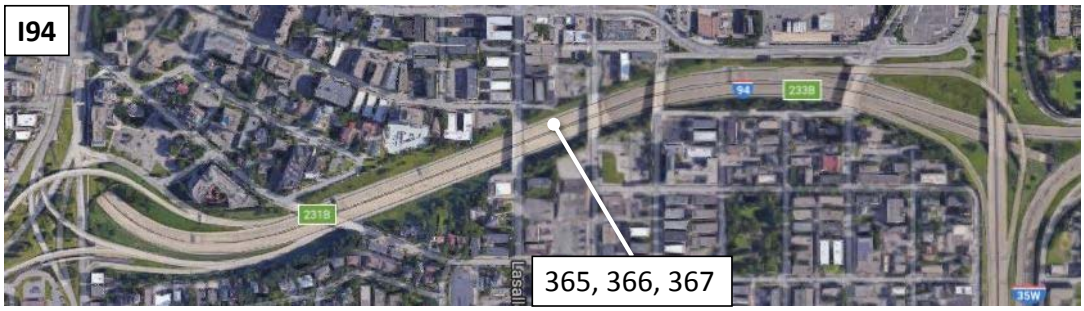


Figure 8.6: Aerial photo of Interstate I94 showing the locations of the selected detector stations. (© Google Maps, 2016)

Figure 8.7: Aerial photo of Interstate TH5 showing the locations of the selected detector station. (© Google Maps, 2016)

8.6.2 Performance measures

Given a classification of a specific data set, certain metrics are taken into account to evaluate the performance of the classifier. Some of the desirable qualities of an AID algorithm are

- a high detection-rate,
- a low false-alarm rate, and
- a short mean detection time.

Unfortunately, these goals are not independent and therefore, there is no perfect AID algorithm which can satisfy all at the same time. An increase in the detection rate or a reduction of the mean detection time is usually accompanied by an increase of the false alarm rate. A high number of false alarms is not only highly undesirable, but can damage the confidence in the detection system. The following four basic ratios are usually used for statistical analysis of classifiers:

- True positives (TP): The number of classifications, where roads that have been predicted to be congested actually are congested.
- True negatives (TN): The number of correct results, where roads have been classified as free flowing and there is no congestion.

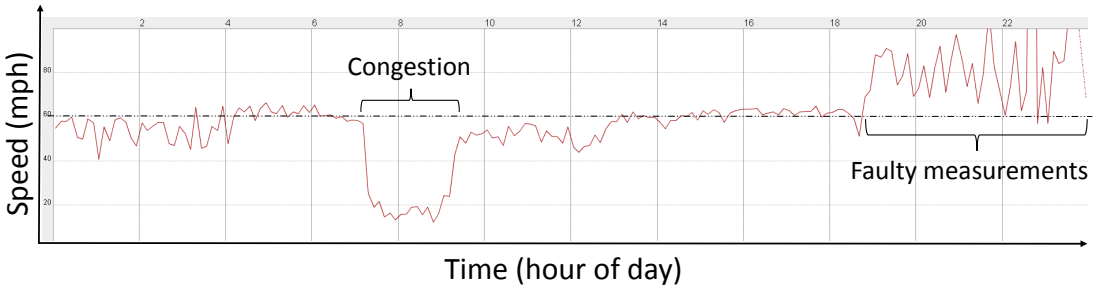


Figure 8.8: Traffic flow profile for arterial I35E, detector station 2447, 2013-06-12. The records contain faulty values from 7 p.m. to midnight.

- False positives (FP): The number of falsely predicted congested roads, while traffic flows freely.
- False negatives (FN): The number of falsely predicted free flowing roads, whereas the road is actually congested.

In other words, TP and TN describe the accuracy of the classifier (the predicted class label matches the actual classification). FP and FN measure the error rate of the evaluated classifier. Naturally, high detection rates and a minimal number of false alarms is desired. However, these two performance measures are not independent. The number of false alarms can easily be reduced by decreasing the sensitivity of the detection algorithm. However, this will result in poor detection rates. In contrast, increasing the detection rate will also increase the false alarm rate. The detection rate is calculate by

$$DR = TP/(TP + FN) \tag{8.2}$$

and the false alarm rate is determined by

$$FAR = FP/(TN + FP) \tag{8.3}$$

8.6.3 Further performance measures

For a comprehensible, statistically evaluation, several metrics have to be considered. As the consideration of only a single metric can be deceiving, another six metrics are used for the evaluation. The accuracy A specifies the number of correct results returned by the classifier as

$$A = \frac{TP + TN}{N} \tag{8.4}$$

where N is the total number of classified situations. A classifier who simply classifies all situations as free flowing achieves high accuracy since the probability that traffic is congested is generally much lower than free flowing traffic. The precision P is calculated as

$$P = \frac{TP}{TP + FP}. \tag{8.5}$$

An algorithm who predicts few or no congestion may result in high precision since the number of FP is minimised. In general, high precision means that the classifier returns more correct than wrong predictions. The recall R measures the proportion of positives that are correctly identified by

$$R = \frac{TP}{TP + FN}. \tag{8.6}$$

High recall can easily be achieved by classifying all situations as congested. The F-measure (or F_1 score) considers both recall R and precision P by

$$F = \frac{2PR}{P + R}. \tag{8.7}$$

Finally, the specificity SP measures the proportion of negatives that are correctly identified by

$$SP = \frac{TN}{FP + TN}. \tag{8.8}$$

In particular, in terms of very skewed classes, a good classifier is depicted by high values for precision and recall (and consequently by a high value of the F-measure).

8.6.4 Parameter study

Selecting the feature vector

Initially, 15 experiments are evaluated, one for each combination of the four traffic variables speed, volume, occupancy, and density. The experimental results are averaged over ten runs, each run executed on a data set monitored by a different detector. Figure 8.9 exemplarily depicts the performance of these feature vectors for the F-measure. In general, the best values are achieved when the feature vector comprises the speed measurements (mostly accompanied by occupancy).

A visualisation of the state space helps to estimate the complexity of the underlying problem. Figure 8.10 depicts a scatter plot showing the dependency between volume and speed (Figure 8.10a), and speed and occupancy (Figure 8.10b) for a random day. The black line visualises a possible separation between states that are categorised as congested or not congested. As it can be seen, the two classes are separated by linear regression.

Better performance can be achieved by reducing or increasing the number of traffic parameters within the feature vector. On the one hand, more features can describe the underlying dynamics of traffic more precisely. On the other hand, a higher number of features expands the search space drastically since data sets are generally sparse in higher dimensions and therefore more data is needed to learn the model. This leads to longer exploration durations while more classifiers are needed to describe the respective feature space. Additionally, smaller sets of features can reduce the variance on the test set, while we may end up overfitting the data with more features. Figure 8.11 shows the dependence between the number of traffic variables in the feature vector and the average number of micro classifiers within the population of the XCSR.

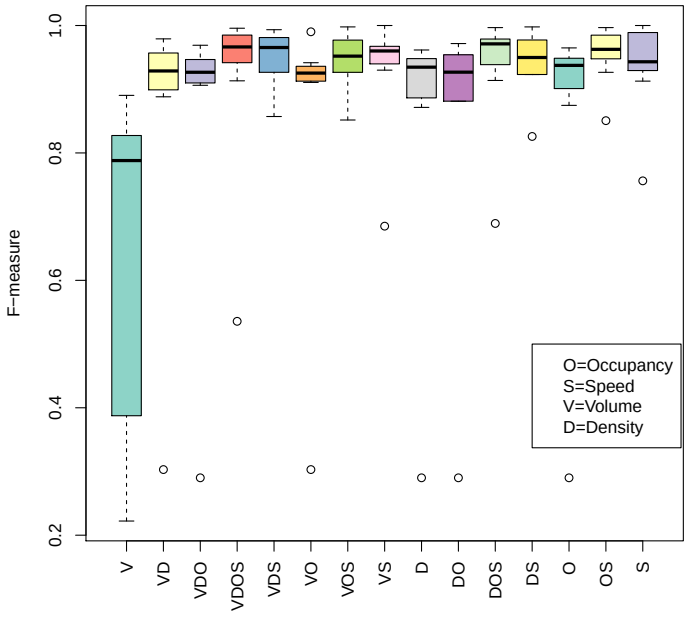
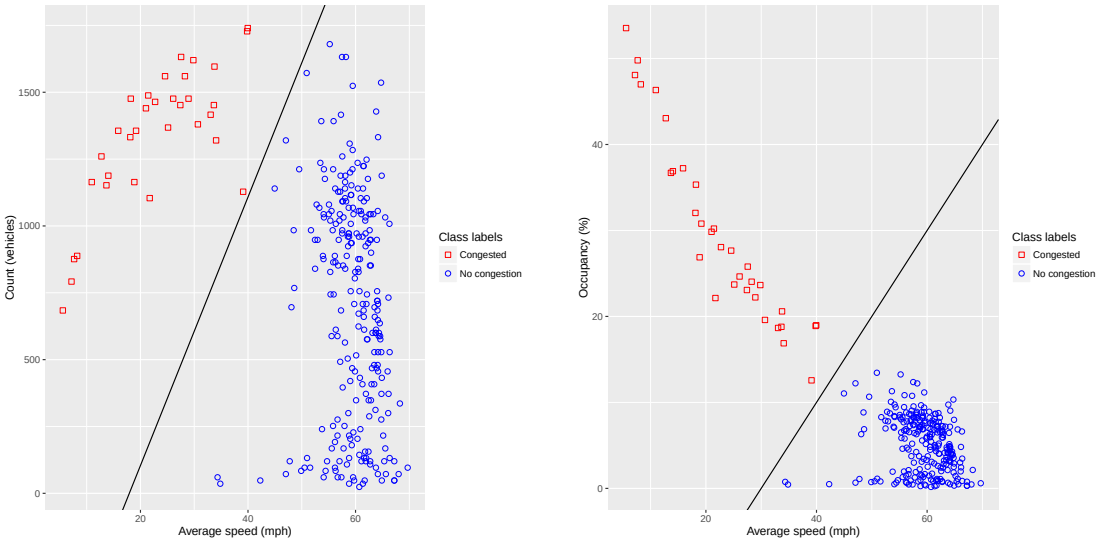


Figure 8.9: Accuracy of several feature vectors evaluated with the F-measure (1.0 equals the best accuracy).

According to [PX06], the two traffic parameters speed and occupancy are chosen about 80% of the time in terms of congestion detection in traffic management centres in the U.S. As can be seen in figure 8.9, the feature vector comprised of these two traffic parameters results in very good performance, thus, further experiments will be conducted with this feature selection. The population size for a feature vector with occupancy and speed was roughly between 3400 and 5400 (see figure 8.11). Therefore, 6000 is chosen as the maximum number of micro-classifiers within the population of XCSR.

Parameter settings for XCSR:

Butz et al. [BW03] summarise the commonly used parameter settings for the learning parameters of XCS. The initial parameter settings are mostly chosen accordingly (Table 8.1). Additionally, a parameter study is conducted for the most important learning parameters: $\beta = \{0.1, 0.2, 0, 5\}$, $\theta_{GA} = \{1, 5, 15, 25, 50\}$, $s_0 = \{0.1, 0.2, 0.5\}$, $m_{cs} = \{0.1, 0.2\}$, $m_{ob} = \{0.1, 0.2, 0.4\}$, $r_{ob} = \{0.1, 0.2, 0.4\}$, $p_X = \{0.2, 0.3, 0.5\}$, $p_M = \{0.04, 0.05, 0.06\}$ and $P_\# = \{0.0, 0.5\}$. The best performance was achieved with the following settings: $\beta = 0.2$, $\theta_{GA} = 5$, $P_\# = 0.0$, $p_X = 0.3$, $p_M = 0.05$. Furthermore, the performance for the unordered bound, the ordered bound, and the centre spread representation for classifier conditions was compared. Unordered bound on average gives the best results with $m_{ob} = 0.2$ and $r_{ob} = 0.2$. An increased learning rate β allows the system to adjust classifiers faster but makes it more sensitive to temporary peaks. The usage of wild-cards in the condition representation is omitted as tests showed worse accuracy. A decrease in the number of executions θ_{GA} of the GA leads to a rising variance



(a) Volume and speed.

(b) Speed and occupancy.

Figure 8.10: Scatter plots showing the state space for different traffic variables measured at highway I35E, detector 2447 (red squares depict congested states, blue circles free flowing conditions).

of results. XCSF is initialised with an empty population. The condition is represented by rectangles.

Parameter settings for SVM:

Concerning the SVM variants, a small parameter study was conducted for C, γ, and the kernel type. A Gaussian kernel is chosen since it is assumed to provide good results when applied to intermediate large data sets with only a few features. The Gaussian chi-squared kernel showed to be computational less expensive than the Gaussian kernel with L2 distance, still providing similar accuracy. It is calculated as

$$k(x,y) = 1 - 2 * \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{x_i + y_i} \tag{8.9}$$

where x_i and y_i represent different samples with n being the sample length. Test runs with $C = \{1, 10, 100, 500, 1000\}$ and $\gamma = \{0.01, 0.1, 0.5, 1\}$ (Figure 8.12 shows the scatter plots with a fitted regression plane) indicate that $C = 10$ and $\gamma = 0.01$ yield good results (F-measures of LaSVM-I=0.89, LaSVM=0.97, SDCA=0.89) without overfitting the model (bias-variance trade-off). For more details on SVMs, the reader is referred to [BHW10].

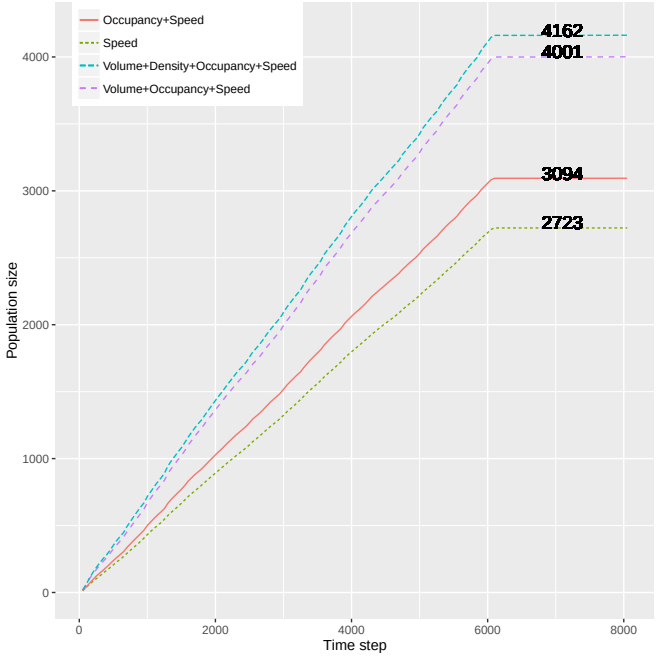
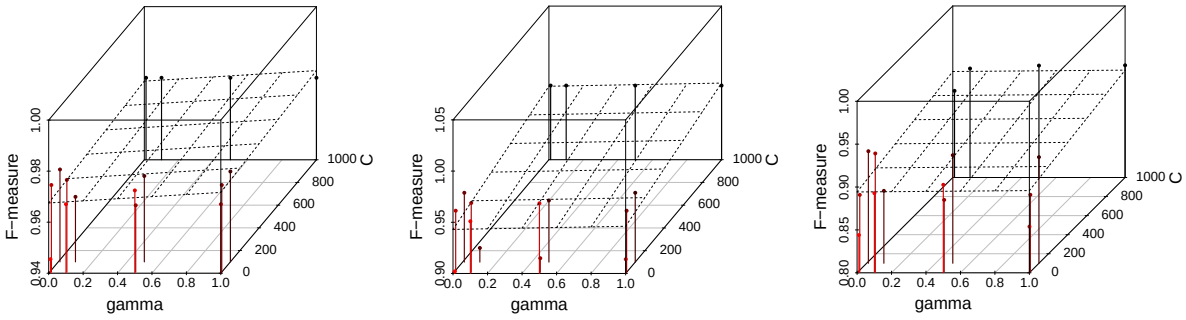


Figure 8.11: The time plot depicts the development of XCSR's average population size for feature vectors with 1, 2, 3, and 4 traffic parameters.



(a) LaSVM **(b)** LaSVM-I **(c)** SDCA

Figure 8.12: Scatter plots with a fitted regression plane showing the F-measure (y-axis) for different parameters of C (z-axis) and gamma (x-axis).

8.6.5 Experimental results

In the following, the results of our evaluation are presented and interpretations are given accordingly. The following results are averaged over all ten data sets from the selected detectors.

Runtime

Table 8.2 shows the mean runtime and standard deviation averaged over ten executions. The evaluation was done on an Intel i7 dual-core with 2.6 GHz and 8 GB RAM. The SVM variants

Table 8.1: Initial parameter settings for the most important learning parameters of XCSR.

N	Max. number of micro-classifiers	800
β	Learning rate for p, ϵ, and ϕ	0.2
ϕ_{init}	Initial classifier fitness	0.01
ϵ_{init}	Initial classifier prediction error	0.0
p_{init}	Initial classifier prediction value	10.0
δ	Class. fitness deletion threshold	0.1
ϵ_0	Classifier accuracy threshold	10
θ_{sub}	Classifier subsumption threshold	20
θ_{del}	Class. experience deletion threshold	20
θ_{GA}	GA application interval	5
p_X	Crossover probability	0.3
p_M	Mutation probability	0.05
α	Fitness reduction factor	0.1
p_{red}	Prediction reduction factor	0.25
s_0	Centre spread factor	0.2
m_{cs}	Mutation prob. for centre spread	0.1
m_{ob}	Mutation prob. for (un)ordered bound	0.2
r_{ob}	Covering prob. for (un)ordered bound	0.5

were used as is from the JKernelMachines framework [PTC13]. Considering execution times, XCSR has a clear benefit over SVM, since it has a much lower runtime, both for training and testing (each data set has 4032 data points). To speed up the on-line and off-line training process of the SVM variants, the number of training epochs E can be reduced. By reducing E from five (the default value in the JKernelMachines framework) to one epoch, the following speed-up can be achieved: LaSVM (8.4 sec.), LaSVM-I (2.6 sec.), SDCA (2.0 sec.).

Table 8.2: Average runtime (and standard deviation) in seconds for one complete experimental run.

	LaSVM	**LaSVM-I**	**SDCA**	**XCSR**
Training (sec.)	14.2 (10.3)	9.5 (14.3)	5.6 (0.5)	2.6 (0.4)
Test (sec.)	0.8 (0.3)	1.7 (1.5)	7.6 (0.2)	0.5 (0.2)

Still, these runtimes have to be interpreted with caution. Each data point of the data set was given one-by-one to XCSR (on-line learning), whereas the SVM's were given the training set as a whole, speeding up the learning process drastically as the model was computed only once. In fact, if the SVMs were trained with on-line learning, adjusting the internal model after every time step, the execution times are significantly longer, for example LaSVM-I needs 12.5 minutes, and SDCA needs 5.5 minutes for a single training run (4032 data points, $E = 5$).

Classification accuracy

Table 8.3 shows the confusion matrix for the test sets. The numbers show that XCSR has a low false alarm rate $FAR = 0.26\%$ and a high detection rate $DR = 95.5\%$ (see equations 8.3 and 8.2). The average number of FP and FN is rather low. The FAR and DR of the SVM variants are as follows: LaSVM ($FAR = 0.21\%$, $DR = 90.0\%$), LaSVM-I ($FAR = 0.43\%$, $DR = 74.3\%$), and SDCA ($FAR = 1.4\%$, $DR = 91.5\%$).

Table 8.3: Confusion matrix reporting the number of true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN) of XCSR, averaged over ten data sets.

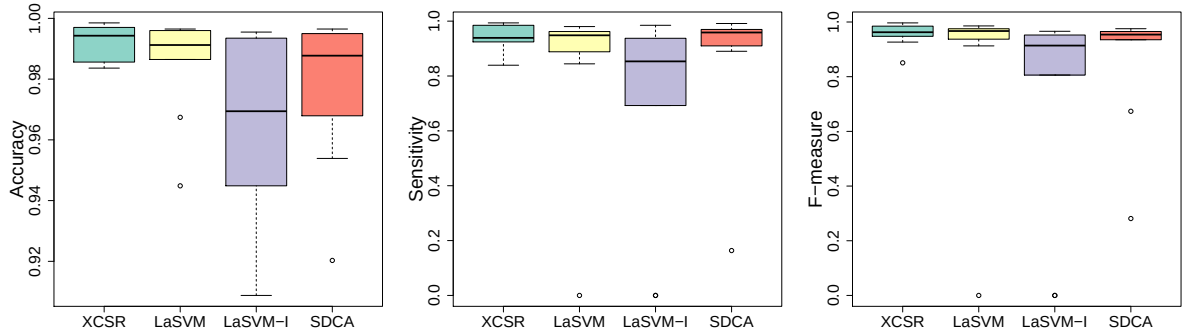
		Actual class		
		Congested	Free flowing	Total
Prediction	Congested	141	5	146
	Free flowing	12	1858	1870
	Total	153	1863	2016

Figure 8.13 presents the results for the before mentioned performance measures. The box plots show the statistical distribution of the average absolute forecast errors. The bottom and top of the box represent the first and third quartiles, and the band inside the box represents the median. Outliers are indicated by separate points.

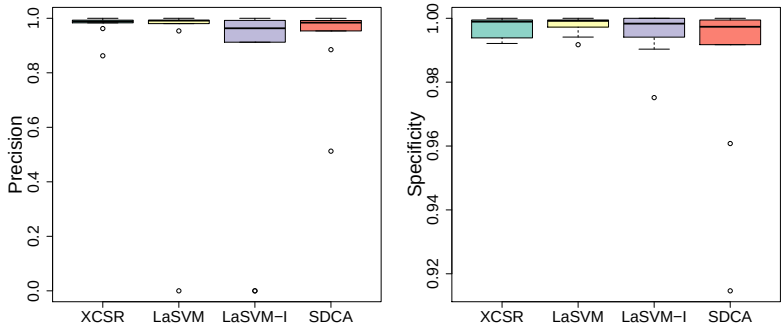
All approaches have high accuracy (an average of 97% and above), classifying most of the congested situations as congested and most of the not congested situations as free flowing. Figure 8.13d indicates that LaSVM-I misclassifies too many situations as congested. In contrast, XCSR classifies most of the not congested situations correctly (see figure 8.13e). Most of the outliers are caused by the TH5 data set which has relatively few congested situations. Deducing from the figures, all SVM classifiers had problems to learn its underlying feature space. Although, XCSR has its lowest values for recall (0.84), precision (0.86), and the F-measure (0.85), its performance is still good. LaSVM and LaSVM-I are not able to learn the task for this data set as they simply classify all situations falsely as not congested. On average, XCSR has better results for accuracy, recall, and F-measure than the SVMs and similar performance for precision and specificity. Concluding, we can be confident that XCSR actually represents a good classifier for this classification problem, also in case of highly skewed classes.

Learning behaviour of XCSR

To get a more direct insight into the learning behaviour of XCSR, the system error (the mean absolute error), the average population size (number of micro-classifiers), and the fraction of correct classifications (percentage of correct classifications in the last k executions) is measured. The system error is calculated as the absolute difference between the actual reward and the system prediction P_a of the selected action, divided by the maximal reward (usually 1000). Figure 8.14 shows the development of the corresponding means. Each data point is the average over 50 successive XCSR executions, averaged over all ten time series. The vertical dotted line marks the end of the training phase after 6000 time steps. The population curve shows the average number of micro-classifiers normalised to the range of $[0; 1]$.



(a) Accuracy. **(b)** Recall. **(c)** F-measure.



(d) Precision. **(e)** Specificity.

Figure 8.13: The box plots show the statistical distribution of the average classification accuracy for XCSR, LASVM, LASVM-I, and SDCA (left to right) considering occupancy and speed.

The plot shows that XCSR is able to improve its performance over time. During testing, XCSR tries to evolve new classifiers, and to explore the state space. Consequently, compared to the testing phase, the system error is higher as XCSR tries different classifications which can be wrong. XCSR applied covering between 16 and 39 times (average: 27). The number of GA executions was between 325 and 777 (average: 494).

Qualitative comparison between XCSR and SVM

The choice between several approaches is often dependent on multiple factors. One aspect is, of course, their performance on historical data which is evaluated and discussed in the following section. Other aspects are the need for complex parameter settings, the interpretability of the internal model by humans, and concerning classification tasks, for example the possibility to add further classes.

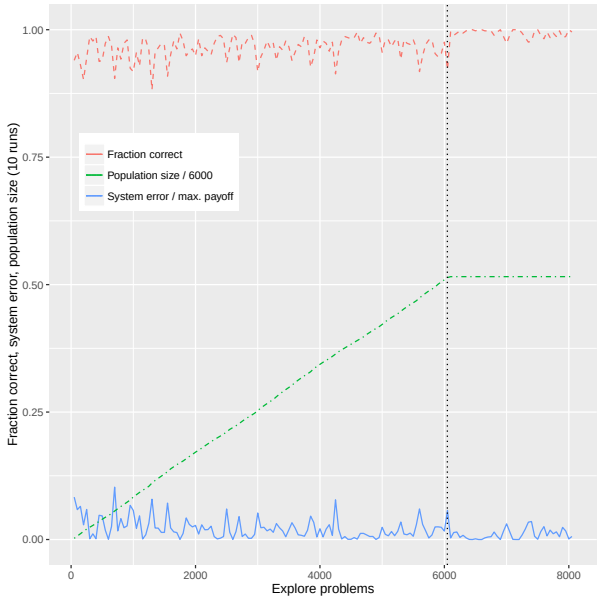


Figure 8.14: Development of the average fraction correct, system error, and population size for XCSR (feature vector: occupancy and speed).

Configuration The convenience to configure the respective technique is important to find the optimal parameter settings in a short amount of time. Mostly, XCSR offers fairly good performance out-off-the-box using the standard parameter settings. Still, a fair amount of parameter studying is needed to find the optimal settings for the most important parameters controlling the learning process. In this aspect, SVM is quite simple to configure since it only requires two hyper-parameters C and γ and the number of epochs E to be set, as well as the kernel to be chosen.

Interpretability Another aspect is the understandability of the internal model. On the one hand, SVMs resemble a very flexible method. Still, their interpretability is very low as the support vectors are difficult to analyse or to visualise [JWHT13]. On the other hand, XCS's internal rule representations are interpretable by humans. Furthermore, the development of the model is still flexible. Classifiers can be added and adapted during runtime, and their respective values, action, and condition are easily understandable.

Number of classes Within this chapter, congestion detection is formulated as a two class learning problem. Still, further classes can be added, for example heavy traffic volume or faulty detector data. Increasing the number of classes is no problem with XCSR, as simply the number of distinct classifier actions have to be increased. Additional classes lead to a more complex learning problem, however, the complexity of XCSR stays the same. In contrast, SVMs are usually two-class classifiers. For multi-class tasks, decomposition methods such as one-against-all or one-against-one are used. Solving a multi-class SVM in one step results in a much larger optimisation problem [HL03].

8.7 Summary

The extended classifier system XCSR was applied to congestion detection on interstates. The evaluation was done with real-world data from inductive loop detectors located in Minneapolis. Compared to three different types of support vector machines, XCSR offers competitive performance. Furthermore, XCSR is significantly faster in terms of runtimes for training and testing. In contrast to the support vector representation of SVM, the rules base of classifiers of XCSR is easily interpretable by humans. In conclusion, it can be noted that XCSR is able to evolve accurate classifiers, offering reliable accuracy and precision for the classification of traffic conditions.

A limitation of this approach is its inability to detect capacity reducing incidents which do not lead to congestion but still influence traffic negatively (such as parking trucks). These events can only be detected by analysing camera feeds (automatically or by manual inspection). These incident alarms can be passed on by the AID component and can be used in the dynamic route guidance system. Sections with incidents are likely to result in higher travel times and can promote further congestion. Therefore, the DRG system should omit route proposals traversing these sections, and instead, propose alternatives (which may be longer but still having less travel time). Consequently, the section is less likely to be proposed by the DRG system. However, sections with are congested due to high-volume traffic are already more likely to be omitted due to the dynamic traffic-dependent estimation of the travel times within the DRG system.

The next chapter transfers XCSR to the distributed congestion detection architecture in OTC. Additionally, the self-optimising adaptation process of the signalisation can be further enhanced by using the congestion alarms raised by the AID component. Extending the green times of outgoing sections, or increasing the cycle time can alleviate congestion at the signalised intersection.

Chapter 9

Distributed urban congestion detection

As summarised in section 8.1, there are several real-world ITS' that monitor the current traffic state. Some of them also provide mechanisms for the autonomous detection of disturbed conditions. Although some use automatic congestion detection algorithms, the congestion management is usually executed by humans. Thus, the next logical step is the integration of these congestion alarms into a self-adaptive traffic management system. This enables an autonomous adaptation of the system to alleviate the negative effects of congestion or even to prevent congestion. So far, comparatively few attempts have been made in this direction. Most research has been limited to algorithms for the detection of traffic congestion.

In contrast to these traffic control systems, the following concept goes one step further, proposing a self-adaptive traffic management process that automatically detects and reacts to congestion in urban road networks. Additionally, the XCSR for congestion detection on highways introduced in section 8.4 is now adapted to urban congestion detection within OTC. In case the current situation is classified as congested, OTC will react with an adaptation of its control strategy. This adaptation can incorporate a modification of the signalisation in terms of green times and cycle time, calculating new route recommendations, or triggering the adaptive progressive signal system mechanism.

The remainder of this chapter is structured as follows: First, an introduction to the distributed automatic congestion detection in OTC is given. Afterwards, XCSR is adapted to work with these concepts and is applied to urban congestion detection. Moving on, an automatic adaptation of signalisation due to congestion within OTC is presented. Finally, the approach is evaluated based on a simulation study.

9.1 Distributed congestion detection in OTC

OTC can be extended with a distributed automatic congestion detection (AID) architecture. *Kleijnowski* [Kle08] presents a distributed architecture based on OTC, and a modified variant of the California algorithm 7 (a static decision tree) for congestion detection in metropolitan areas.

The concept works as follows. Each standard OTC controller is extended by an additional module for the real-time automatic congestion detection, designed for application in urban areas (Figure 9.1). This approach is completely distributed and works on local knowledge only. Based on locally monitored detector data, each TLC is enabled to detect nearby congestion. Therefore, no centralised server is needed to collect data, but the communication infrastructure can be

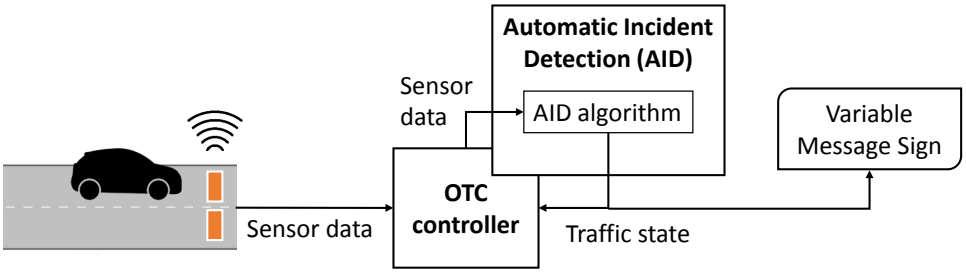


Figure 9.1: Distributed congestion detection using locally monitored sensor data.

used to disseminate information to nearby TLCs. This allows for an incorporation of additional data, such as road conditions and traffic states from nearby intersections. Inductive loop detectors serve as detector stations, monitoring the current traffic flow at the local intersection and its approaching sections. The AID component is executed in cycles. Every cycle, an automated analysis of the up-to-date sensor data is performed by congestion detection algorithms in order to classify the current traffic conditions into congestion-free or congestion-bound.

9.1.1 Architecture and prerequisites

Figure 9.2 depicts the architectural prerequisites for the installation. At each signalised intersection, an AID component extends the standard OTC controller (Figure 9.1). It is responsible for one or more *monitoring zones* (MZ) in its direct vicinity. A MZ represents the area where the respective AID component applies an AID algorithm. The separation of the road network into several MZ's splits the complex problem of a network-wide congestion detection into feasible sub-problems which are handled by individual AID components. Each component can manage multiple MZs, whereas each MZ contains certain sections which are monitored by detector stations.

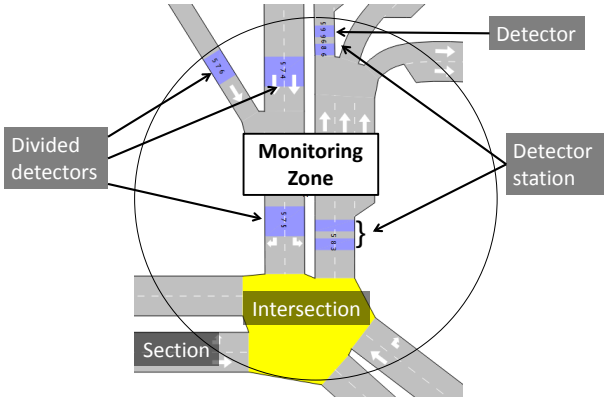


Figure 9.2: Distributed congestion detection by mapping detector stations to monitoring zones.

First, at least two *detector stations* are necessary to monitor the upstream and downstream traffic conditions on a section. A *detector station* usually consists of two simple *detectors*. Second, each zone can be assigned a different automatic congestion detection (AID) algorithm. A MZ can also cover additional branching sections via *divided detector stations* which helps to monitor

complex traffic patterns more precisely. Thereby, the complex patterns created by incoming and outgoing traffic streams in terms of branching sections are taken into account. Finally, a detection algorithm is assigned to each monitoring zone. The traffic parameters monitored by these detectors (such as inductive loop detectors) can be utilised by the assigned AID algorithm.

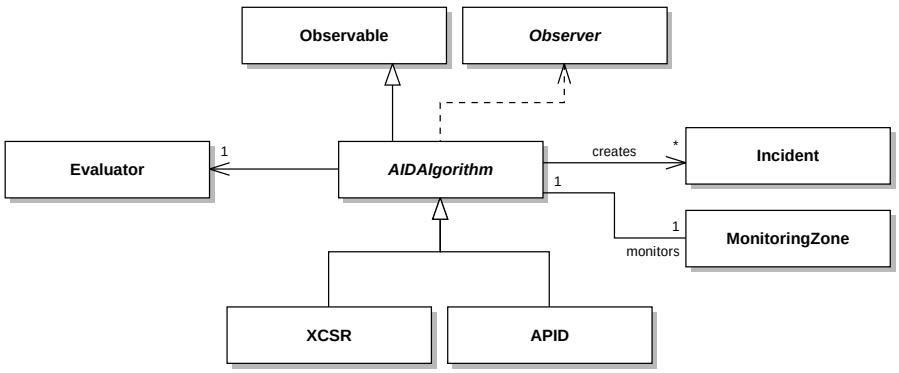


Figure 9.3: Class diagram showing the AID algorithm interface.

OTC demands no specialised AID algorithm and the applied algorithms and their parametrisation can be different for each zone. The applicable algorithms can range from simple heuristics to sophisticated machine learning techniques. However, each algorithm has to implement an abstract class defining certain interfaces. The class diagram in figure 9.3 shows this interface and its related entities. Each *AIDAlgorithm* monitors exactly one monitoring zone. It can create several *Incident* objects which are also used to evaluate its performance within the *Evaluator*. Concrete implementations of congestion detection algorithms, such as XCSR or the all-purpose incident detection algorithm (APID), have to implement the abstract interface. The algorithms are notified about new detector values by implementing the observer pattern. For easy integration into a graphical user interface and for additional monitoring, they also extend *Observable*, thereby following the model-view paradigm.

9.1.2 Automatic reaction to congestion alarms

In previous work [Kle08], the architecture is presented and one algorithm was implemented. In this work, we propose extensions to this initial architecture.

Let us summarise the process again: The controller receives traffic data from nearby sensors describing the traffic states at nearby sections. Afterwards, this data is used by an automatic congestion detection algorithm to classify the current traffic conditions into congestion-free or congested. The initial framework proposed by Klejnowski just collects these congestion alarms. However, a reaction process is not presented.

This thesis proposes new approaches for how the OTC system can react automatically to these alarms. In case the selected algorithm classifies the current traffic situation as congested, OTC will react with an adaptation of its control strategy. This adaptation can incorporate 1) a modification of the signalisation in terms of green times, 2) a modification of the proposed route recommendations, or 3) triggering the adaptive progressive signal system mechanism. Concerning the first point, we want to reduce the inflow to the congested section. This is achieved by

reducing green times towards this section. For the second point, it is assumed that the routing protocols are already able to detect and to react to the detection of congested sections by proposing alternative routes. However, it can be beneficial to integrate information about incidents that do not induce congestion but may influence traffic negatively (for example a parking truck reduces the capacity of the section). For the implementation of the third point, we just have to make a function call to trigger the progressive signal algorithm in case a congestion is detected. Section 9.3 of this chapter presents how the first idea can be implemented.

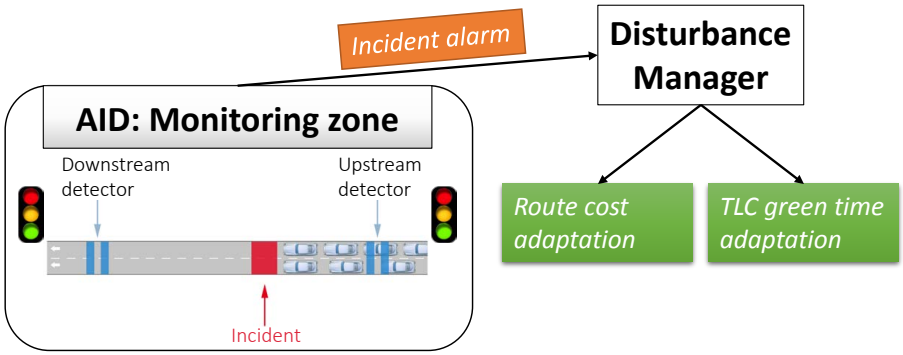


Figure 9.4: Automatic detection of a congestion by two divided detector stations and the automatic reaction process within OTC.

Figure 9.4 depicts the proposed process in case of a congestion alarm. First, an alarm is raised by an AID algorithm responsible for the according monitoring zone. Next, the alarm is passed-on to the *disturbance manager*. A disturbance is defined by the corresponding congestion causing the disturbance, its start time, its end time, its severity, and its exact location. The disturbance manager automatically estimates the severity of the congestion by using sensor data from traffic detectors and approximation heuristics. The average speed sp and the average detector occupancy occ can serve as measures for the congestion level of the section. Based on these two characteristics the severity of a congestion on a section can be estimated by

$$severity = sev_{sp} * sev_{occ} \tag{9.1}$$

with

$$sev_{sp} = 1 - min(1, \frac{\sum^{d_i} v_i}{d_i} / v_{max}) \tag{9.2}$$

and

$$sev_{occ} = min(1, max(occ_i)/100). \tag{9.3}$$

where v_{max} is the speed limit, d_i denotes the detector station on the section, where v_i resembles the speed monitored by d_i, respectively occ_i the occupancy of d_i over a certain time span. Therefore, this factor lies between 0 and 1 whereas 1 denotes the highest severity. The severity factor then serves as additional parameter for the adaptation of green times at the signalised intersection and as a penalty factor within the travel time estimation on the congested section. An AID algorithm can take several states:

- Congestion free: Traffic is free flowing.
- Tentative congestion alarm: Congestion is assumed, but no congestion alarm is raised yet.

- Congestion confirmed: The tentative congestion was confirmed by a subsequent test. A congestion alarm is set off.
- Congestion continuing: The detected congestion is still present.

Depending on the chosen AID algorithm, there can be additional states. To reduce the number of false alarms, an additional verification step by a traffic engineer can be executed (i.e. by examining live feeds from CCTV cameras).

9.2 XCSR for urban congestion detection

In the following, we adapt XCSR to be used as an AID algorithm within the AID component (as depicted in figure 9.1). The traffic sensors output their values to OTC in fixed cycles. This data is passed forward to the AID component where it is aggregated over a certain time span, and serves as feature vector for the XCSR.

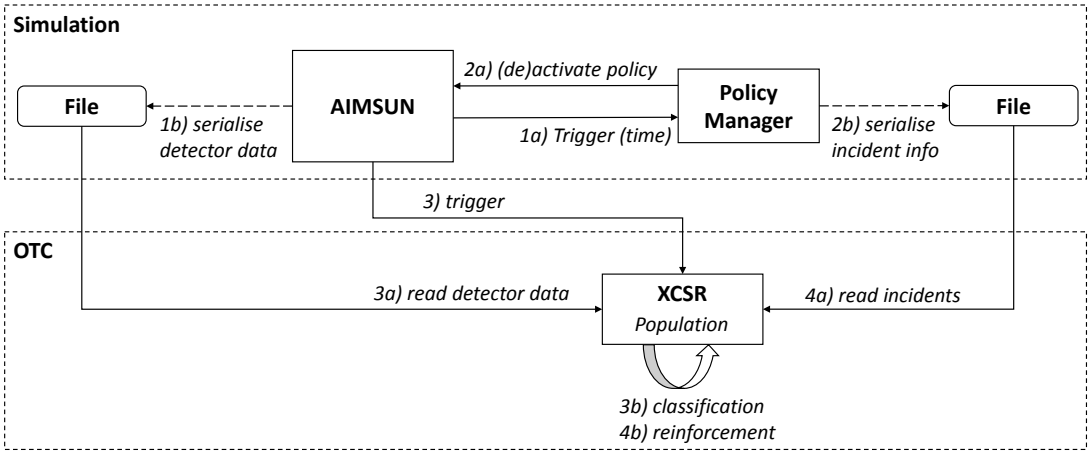


Figure 9.5: The simulation-execution workflow.

Figure 9.5 depicts the setup for the simulation and evaluation of XCSR within OTC. Again, we use Aimsun to simulate the traffic network and to trigger any events throughout the simulation. Aimsun's API does not allow to create random policies, such as lane closures or blockages, at runtime out-off-the-box. Therefore, several policies in Aimsun are created by hand that are located at different locations in the evaluated area. These predefined policies can then be triggered via external scripts. Python scripts are used to activate and deactivate policies during a simulation run. This gives us the freedom to randomly start and end policies with different random durations. Additionally, the start and end times and the location of each triggered policy is exported to log files.

At every time step, Aimsun executes the python scripts and the *Policy Manager* potentially activates or deactivates a policy and the executed policy is written to a log file. In addition, Aimsun also calls OTC and the XCSR is triggered to evaluate the received sensor values. To be more precise, in every time step, the last monitored sensor values from two detector stations - forming a monitoring zone - are used to build the input feature vector. The according classification result

is compared to the actual traffic state which can be received from the stored log files. Finally, the rules from XCSR's population being responsible for the given classification are reinforced in dependence of its correctness.

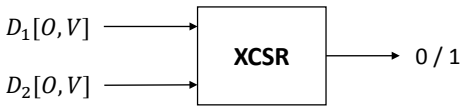


Figure 9.6: XCSR classifies the input (occupancy and velocity) received from two detector stations.

The input of XCSR is defined by the values received from two detector stations (forming a monitoring zone). Again, we chose occupancy and speed as input for our model. Afterwards, XCSR classifies the resulting feature vector into congested (1) or uncongested (0).

9.2.1 Simulation study: Urban road network

To evaluate the performance of the adapted XCSR, this approach is compared to a well-established algorithm for congestion detection, the All Purpose Incident Detection (APID) algorithm [MW91, DCG12] (see section 8.2). It represents a static decision tree algorithm extending the California algorithm 7 with additional states and detection routines, such as for light and medium traffic. However, it needs a fair amount of parameter tuning before it can be reliably applied to a certain section.

A simulation model of a real-world network with eleven intersections located in Hamburg, Germany (Figure 9.7) was developed. The parameter study and the evaluation are done with Aimsun 8 [BC02], a professional traffic modelling and simulation software.

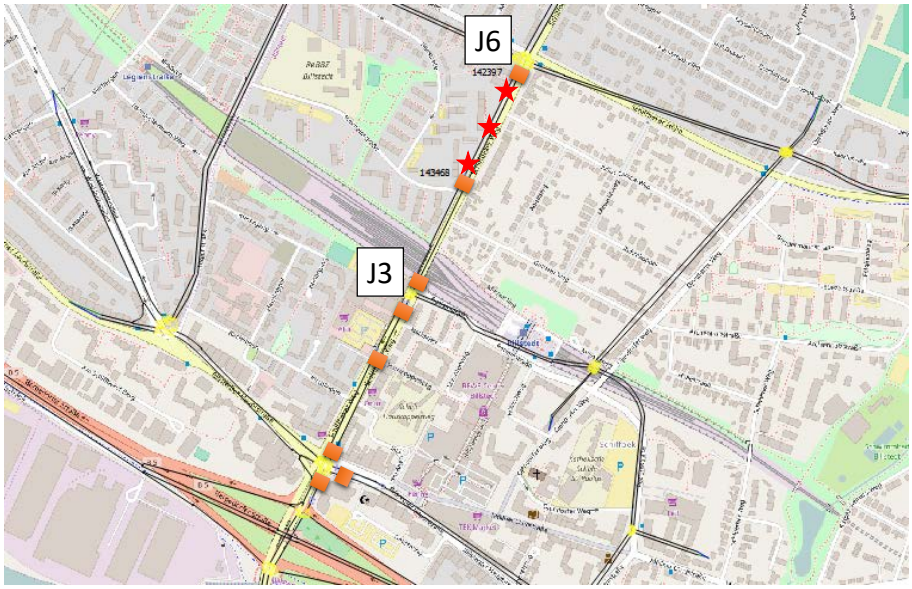


Figure 9.7: Aimsun simulation model of Billstedt in Hamburg, Germany. Stars highlight locations where incidents are created, rectangles mark locations of detector stations.

Parameter study

To find a suitable parametrisation for XCSR and APID, an initial parameter study was executed. The results are averaged over ten random replications by means of ten-fold cross-validation and visualised as violin plots. These plots are similar to box plots, but also show the probability density of the data. They include markers for the median and the first and third quartile of the data. Every violin plot depicts the results of a different configuration which parameter setting is shown at the bottom of the plot.

The APID algorithm is configured in accordance with [MW91] and [DCG12] (see table 9.1). APID is executed in intervals of 40 seconds.

Table 9.1: Parameters settings of the APID algorithm.

Control parameter	**Value**
Compression wave test	disabled
Persistence test	enabled
Medium traffic congestion detection	enabled
Compression wave test period	50 sec.
Persistence test period	50 sec.
Medium traffic flow threshold	80
Congestion clearance threshold	-0.4
Persistence test threshold	0.4
Compression wave test threshold 1	-1.3
Compression wave test threshold 2	-1.5
Congestion detection threshold 1	80
Congestion detection threshold 2	0
Congestion detection threshold 3	20.8
Medium traffic congestion threshold 1	0.0
Medium traffic congestion threshold 2	0.0

Figure 9.8 shows the performance of XCSR, as expressed through the F-measure, for different parameter settings. From bottom to top, the parameters given are the learning rate β, the activation interval of the genetic algorithm, the prediction error reduction constant, the mutation rate and the crossover rate. The violin plots show that a higher number of classifiers increases the average precision. Furthermore, a higher mutation rate also improves the performance. However, an increase in the number of classifier eventually leads to an increase in the computational complexity, therefore, increasing the runtime of XCSR.

Additionally, figure 9.9 shows how a learning rate of $\beta = 0.2$ reduces the system error of XCSR compared to a lower learning rate of $\beta = 0.1$. This can also lead to a speed-up in the learning process. In this plot, the lowermost parameter is the population size instead of the learning rate.

On average, the best outcome was reached by using the configuration as summarised in table 9.2.

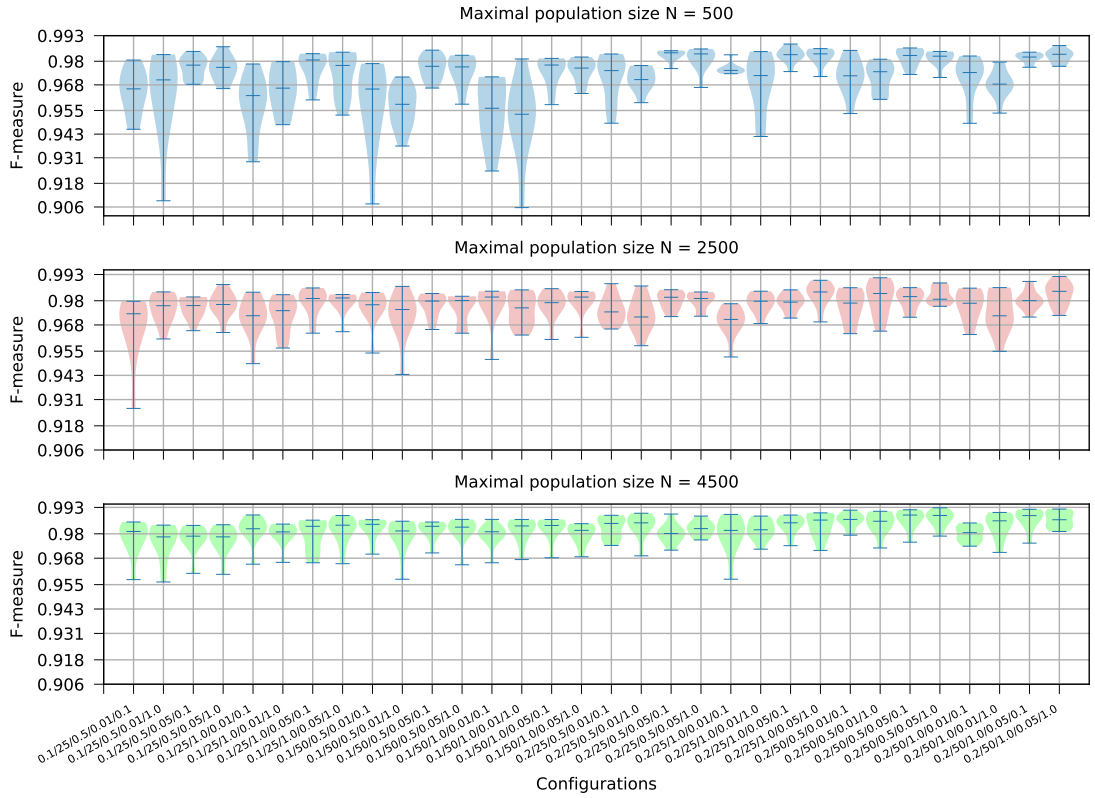


Figure 9.8: Performance of XCSR for different parameter settings in dependence of the population size, evaluated by the F-measure (higher is better).

Table 9.2: Best parameter setting as found by a parameter study for XCSR.

Control parameter	Value
Max. population size N	4500
Learning rate β	0.2
GA execution interval θ_{GA}	50
Crossover probability p_X	0.5
Mutation probability p_M	0.05
Prediction error reduction	1.0

Experimental setup

For further evaluation, the simulation model depicted in figure 9.7 is used. The simulation duration is eight hours. It simulates different traffic conditions, such as rush hour and low traffic. Figure 9.10 depicts the total number of trips throughout the simulation period. The congestion-inducing policies are created randomly in random intervals. The detector values are averaged over a time span of 40 seconds.

As can be seen in figure 9.11, one- or two-lane-blocking events can occur at different locations between the two detector stations. The distance between these two detector stations is 280m.

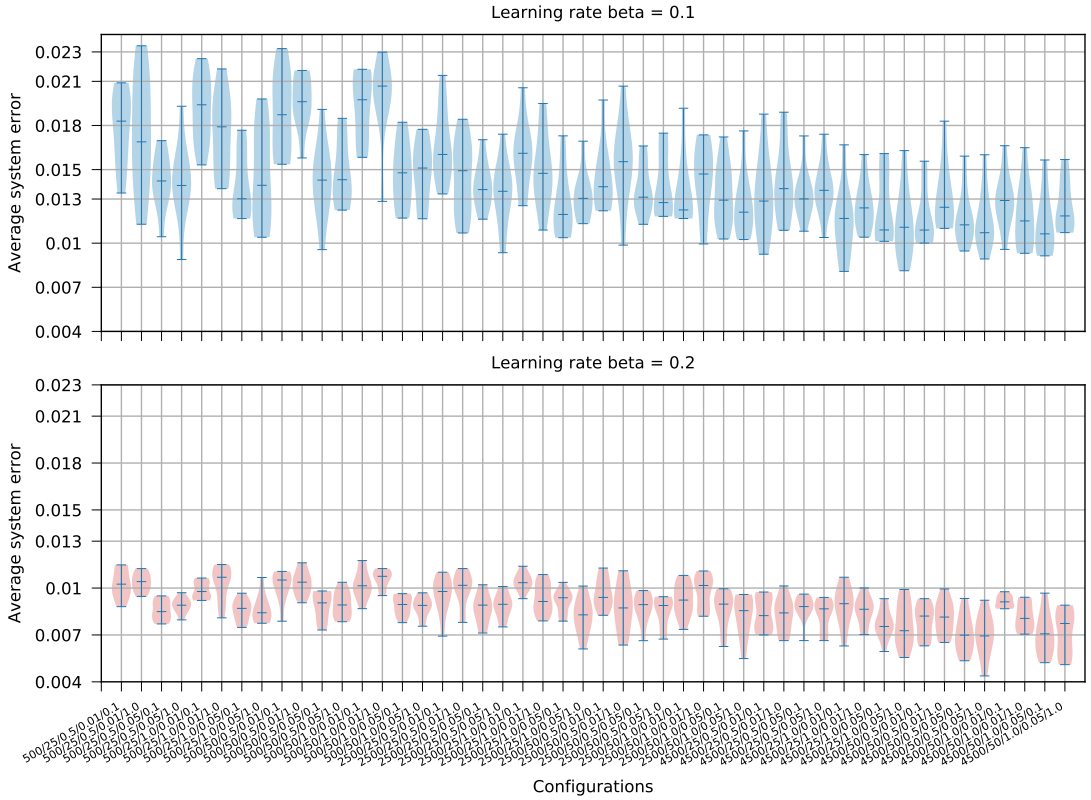


Figure 9.9: Average system error of XCSR for different parameter settings in dependence of the learning rate β, evaluated by the F-measure (lower is better).

The distance from detector station 1 to the first incident location and from detector station 2 to the third incident location is both 10 meters. The second incident location is located in the middle of the two detector stations.

The duration of an incident is a randomly generated time between six and 35 minutes. The length of an incident ranges from a few meters (for example simulating a parking car) to 50 meters (such as a construction site). This setting was chosen since the detection rate is dependent on the location of the congestion, relative to the detector stations. There is a rest period of ten to 25 minutes between two incidents.

Experimental results

To compare XCSR against APID, we use several performance measures which are introduced in section 8.6.2 and section 8.6.3 in the previous chapter. These measures calculate values for false alarm rates, detection rates, and several measures which are typically used to evaluate classifiers.

The experimental setup is as follows. Both evaluated methods receive the same input values from two detector stations located on the simulated section. These are occupancy and speed values. XCSR's population is based on a classifier set that was generated during independent

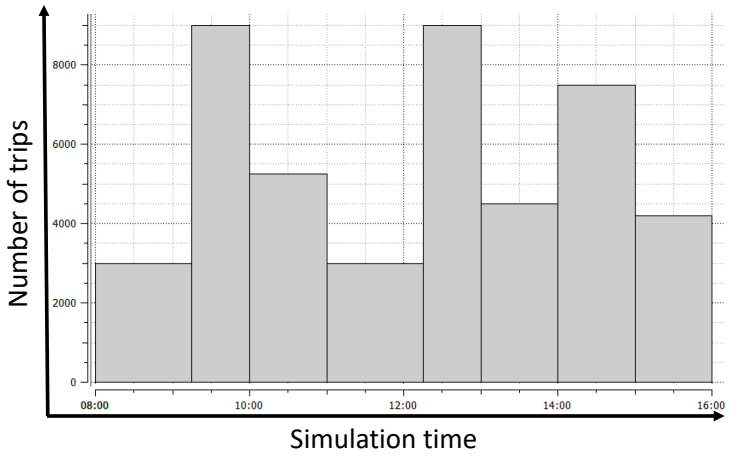


Figure 9.10: The absolute number of trips throughout the simulation period.

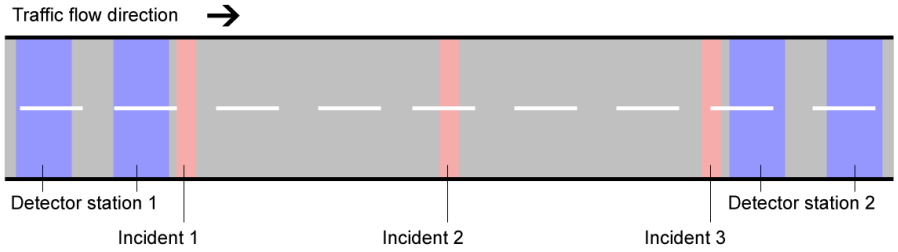


Figure 9.11: Incident are created at three different locations between the two detector stations.

test runs and annotated by hand. During evaluation, XCSR is not further trained but relies on this imported rule set. We chose this setup since a supervised training process at runtime would require a traffic expert to evaluate every output given by XCSR. To be more precise, the traffic classification by XCSR would be evaluated by a traffic engineer. The engineer would specify the actual traffic state as reward for the reinforcement of XCSR's classifiers. In a real scenario, this would be too expensive and time-consuming.

Table 9.3: Results classified by algorithm and incident type.

Algorithm	Incident type	Accuracy	Recall	Specificity	Precision	FAR (%)
APID	One lane	0.844	0.000	1.000	0.000	0.0
APID	Two lanes	0.750	0.877	0.712	0.523	25.8
APID	One and two lanes	0.819	0.706	0.858	0.626	14.2
XCSR	One lane	0.875	0.637	0.925	0.670	7.5
XCSR	Two lanes	0.948	0.882	0.973	0.934	2.7
XCSR	One and two lanes	0.922	0.789	0.957	0.839	4.3

Figure 9.3 summarises the results. As we can see, APID is rather reliable in detecting section-blocking two-lane incidents. Confusion matrix 9.5 backs this observation. Still, APID classifies congested conditions as free flowing too often. APID has problems detecting single-lane incidents where the traffic flow is not substantially different from free-flowing conditions. This is

also depicted in confusion matrix 9.4. This results in a recall value of zero, meaning that APID classifies every sensor value as not congested. Interestingly, APID can also not deal with congestion not being located between two detector stations. APID is unable to detect any congestion events located before the first or after the second detector station.

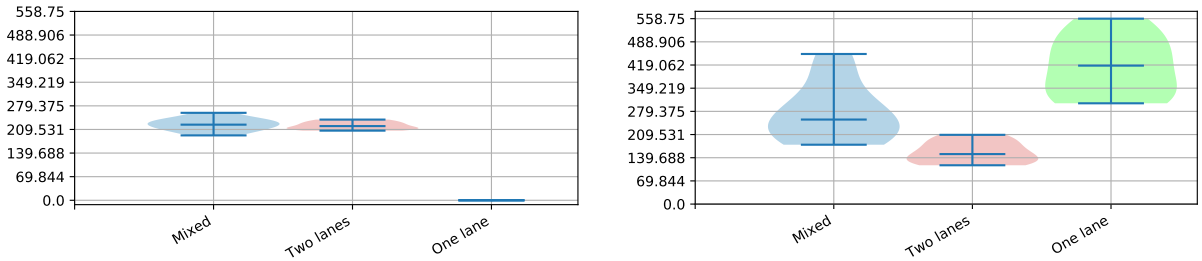
Table 9.4: Confusion matrix with true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN) for APID. Averaged over ten simulation runs creating single-lane incidents only.

		Actual class		
		Congested	Free flowing	Total
Prediction	Congested	0	0	0
	Free flowing	112	607	719
	Total	112	607	719

Table 9.5: Confusion matrix with true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN) for APID. Averaged over ten simulation runs creating two-lane incidents only.

		Actual class		
		Congested	Free flowing	Total
Prediction	Congested	185	128	313
	Free flowing	25	357	382
	Total	210	485	695

As APID has an additional verification step before an alarm is actually raised, the number of classifications can vary. This is also the reason why the number of APID's classifications is lower than XCSR's. On average, the mean time to detection (MTTD) is approximately 224 seconds for two-lane incidents (figure 9.12).



(a) APID **(b)** XCSR

Figure 9.12: Violin plots showing the mean time to detection (MTTD) for one-lane, two-lane, and mixed incident scenarios.

In contrast, XCSR is able to reliably detect one-lane and two-lane incidents. XCSR classifies most situations correctly while also having a low false alarm rate. On average it is lower than five percent. Compared to two-lane incidents, one-lane incidents are a little harder for XCSR to classify correctly. The mean time to detection is approximately 156 seconds for two-lane incidents, and 414 seconds for one-lane incidents. In conclusion, the mean time to detection and the false alarm rate are higher for single lane incidents. This attributes to the fact that the

impact is lower than road blocking congestion. However, the evaluation shows that XCSR can be applied in both scenarios.

Table 9.6: Confusion matrix with true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN) for XCSR. Averaged over ten simulation runs creating single-lane incidents only.

		Actual class		
		Congested	Free flowing	Total
Prediction	Congested	95	42	137
	Free flowing	48	535	583
	Total	143	577	720

Table 9.7: Confusion matrix with true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN) for XCSR. Averaged over ten simulation runs creating two-lane incidents only.

		Actual class		
		Congested	Free flowing	Total
Prediction	Congested	206	13	219
	Free flowing	25	476	501
	Total	231	489	720

Summary

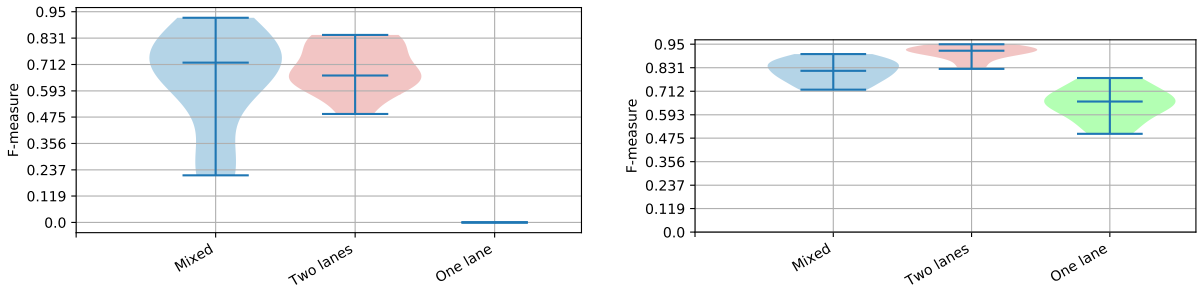
In this chapter, the learning classifier system XCSR was adapted to congestion detection in urban areas. First, a workflow was introduced to couple XCSR with the traffic simulator Aimsun. Thereby, incidents can be created at random at runtime. Every 40 seconds, XCSR is executed to evaluate the current traffic conditions. Afterwards, its classification results are compared to the actual incident alarms.

The evaluation results show that XCSR provides better results compared to a well-established algorithm for congestion detection, the all-purpose incident detection algorithm. In summary, single lane incidents are more difficult to detect than two-lane incidents, both for APID and XCSR. However, APID almost never recognizes any one-lane incidents, whereas XCSR offers good results even for this more complex problem.

Figure 9.13 summarises the results. The violin plots depict the statistical distribution of the F-measure over all evaluation runs. As we can see, XCSR has higher, therefore better, mean values for the F-measure. The deviation is also less broad, compared to APID.

9.3 Automatic adaptation of signalisation based on congestion alarms

To alleviate the negative effect of congestion, traffic streams have to be relocated to other streets, decreasing the traffic flow near the problematic area. This can be achieved by reducing the



(a) APID **(b)** XCSR

Figure 9.13: Violin plots showing the F-measure for one-lane, two-lane, and mixed incident scenarios.

green times of traffic lights entering the congested road or by increasing green times at outgoing sections. The automatic adaptation of green times involves the following steps:

1. Localize the exact location of the congestion.
2. Determine the signal groups sg of the incoming turnings affecting the congested road.
3. The green time of the phases in sg will be reduced in the following. All other phases p are marked for extension of their green times.
4. Filter out interphases and phases in sg with green times below the minimal green period.
5. Calculate the available amount of green time reduction of the phases left in sg.
6. Shorten the green times of the phases in sg.
7. Equally increase the green time of the signal phases in p.

Note that this algorithm ensures that the cycle time of these signal plans stays the same as before. However, it can be easily adjusted to extend the cycle time, for example by giving more green time to signal groups not belonging to the congested road. The phase lengths are always in whole numbers. Furthermore, no conflicts between different signal phases are created, since the relationship between phases and signal groups stays unchanged. The interphases stay the same in terms of their duration, and in terms of their position within the signal plan. According to the German handbook for traffic control (RiLSA) [fSuVAVuV10], the recommended minimal duration for phases is five seconds. Therefore, phases with green times less or equal to this duration are not shortened further.

For n phases (excluding interphases), the maximal amount of green time reduction is calculated as

$$t_{change} = \lfloor \lambda * \sum_{i=1}^{n} t_{shorten}[i] \rfloor \tag{9.4}$$

where $t_{shorten}[i]$ is calculated as

$$t_{shorten}[i] = \begin{cases} p_{GT}[i] - t_{red} & \text{if p can be shortened} \\ 0 & \text{otherwise} \end{cases} \tag{9.5}$$

To control the extend of the algorithm, a green time reduction factor λ defines how much green time can be transferred from the shortened phases to other phases. Additionally, an adaptation threshold t_{red} can define the maximally allowed change of green time (for example 5 seconds). In case that $t_{change} = 0$ (no green time reduction possible) or that the number of phases that can be extended is zero, the algorithms terminates and the active signal plan stays unchanged. Of course, this procedure can also be applied in case there is more than one congested section. The adapted signal plan stays active as long as the applied AID algorithm confirms that the congestion is still ongoing. Afterwards, the signal plan which was active before the change, is restored.

9.3.1 Case Study: Isolated intersection

The following case study explains the introduced algorithm using the example of a real-world intersection. Figure 9.14 shows the model of a three-armed intersection at the "Rotes Tor' in Augsburg, Germany. The signal plan (figure 9.15) and the site map were provided by the civil engineering department. The intersection has six signal groups, two for each incoming section. The signal plans is divided into eleven phases with a cycle time of 90 seconds, two of them being interphases (phases 6 and 8) (Table 9.8).

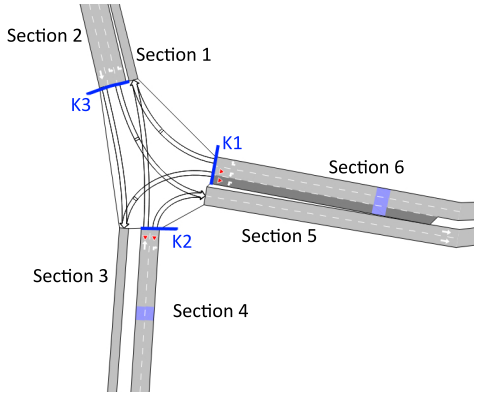


Figure 9.14: Simulation scenario with a single intersection located at the "Rotes Tor" at Augsburg, Germany.

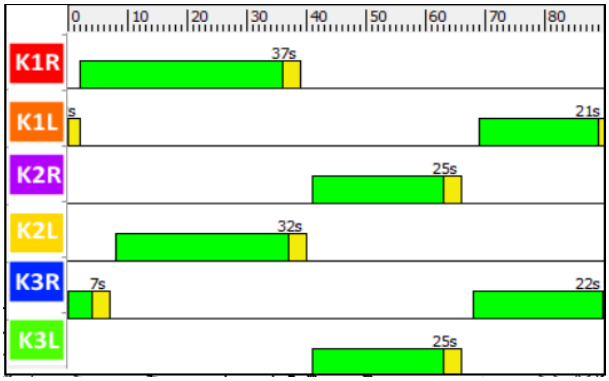


Figure 9.15: Signal plan with six signal groups and a cycle time of 90 seconds.

Let's assume a congestion occurs on section 1. Its incoming traffic flow has to be limited to mitigate the building of congestion. The signal groups and their according phases providing traffic to section 1 are K1R (2, 3, 4) and K2L (7). Consequently, their green times have to be limited to reduce the inflow to the congestion location. Phases 2, and 3 are already shorter than the minimal green time of five seconds, therefore, we won't shorten them. As a consequence, only phases 4 and 7 are available for green time reduction and five phases can be extended (1, 5, 9, 10, and 11). If the durations of phases 4 and 7 are reduced to the minimal green time of five seconds, 23 and 17 seconds are maximally available for green time extension of the other phases. Assuming $\lambda = 0.25$, 6 and 4 seconds (rounded off to the next full second) can be can transferred to the phases 1, 5, 9, 10, and 11. Therefore, each of these five phases is extended by two seconds. Table 9.8 depicts the green times before and after the green time adaptation process.

Table 9.8: Signal plan before and after the green time adaptation ($\lambda = 0.25$). Interphases are highlighted in grey, shortened phases in red, extended phases in green.

Phase	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	$\sum$
old	2	2	4	28	1	4	22	5	1	20	1	90
new	4	2	4	23	3	4	17	5	3	22	2	90

In the following, the evaluation scenario and the experimental results based on a real-world road network (Figure 9.7) are presented. The results of a simulation scenario are compared with and without the adaptation of the signalisation based on congestion alarms.

9.3.2 Simulation study: Road network

After 25, 55, and 85 minutes, a congestion builds up on the section going from intersection J6 to J3. Therefore, during incidents the signal plan at J6 (Figure 9.16) is changed in the way that less green time is given to turnings going towards J3.

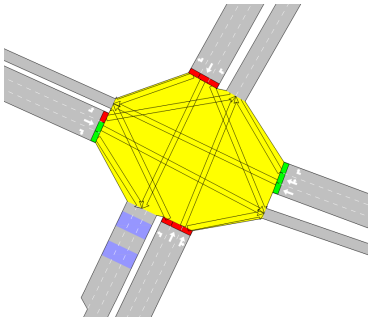


Figure 9.16: Simulation model of intersection J6.

As mentioned by [DCG12], the location of the congestion can influence the detection rate. Thus, the incidents are located at three different locations within this monitoring zone. The incidents block both lanes and last fifteen minutes. Detector stations are installed approximately ten meters after the beginning, ten meters away from the ending, and in the middle of this section.

The signal changes and phase durations simulated within Aimsun are controlled externally by OTC. For the eleven signalised intersections, the fixed-time signal plans have a common cycle time of 90 seconds with two to four phases. The duration for yellow light is two to three seconds. The simulations were executed on an Intel Core i7 quad-core CPU with 2.6 GHz and 8 GB RAM.

9.3.3 Experimental results

The following section presents the results of the evaluation executed as described before. Every second, APID receives its input from traffic detectors simulated by Aimsun. The traffic data (speed and occupancy values) is averaged over a period equal to the execution interval. Initially, different execution intervals of 30, 60, and 120 seconds are evaluated. The according results are given in table 9.9. Interestingly, an execution interval of 120 seconds results in a low MTTD, however APID did not recognize the second congestion. Since an execution interval of 30 seconds results in the lowest MTTD, this value is used for the following experiments.

Table 9.9: Mean time to detection (MTTD) in seconds for execution intervals of 30, 60, and 120 seconds.

Start - End	**30**	**60**	**120**
25 - 40	32m 40s	33m 20s	32m 0s
55 - 75	63m 20s	62m 40s	not detected
85 - 100	91m 20s	96m 0s	93m 20s
MTTD	7m 27s	9m 0s	-

To evaluate the adaptation strategy of OTC considering congestion alarms, a deeper look is taken into the signal plans created. Table 9.10 summarises the adapted green times for different values of λ. Again, $\lambda = 0$ displays the standard OTC strategy. For $\lambda = 1$, the cycle time is slightly increased by three seconds as the reduction of phase 7 would have resulted in a value below the minimal allowed green time. Therefore, it is set to the minimal duration of five seconds.

Table 9.10: Intersection's J6 adapted green times in seconds for $\lambda = \{0, 0.25, 0.5, 0.75, 1\}$. Interphases are highlighted in grey, shortened phases in red, extended phases in green. The duration of yellow light is three seconds.

λ	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	10	4	30	4	10	4	24	4
0.25	15	4	25	4	15	4	19	4
0.5	21	4	19	4	21	4	13	4
0.75	26	4	14	4	26	4	8	4
1	32	4	8	4	32	4	5 (2)	4

In the following, the presented adaptation strategy of the signal timings is compared against the standard OTC system running a fixed-time signalisation plan as introduced in section 2.2. Each approach is evaluated based on the following performance criteria:

- The total travel times for the complete network in seconds per kilometre,

- the average delay times (for example due to red traffic lights) in seconds per kilometre,
- the average stop time for the complete network in seconds per kilometre,
- and the vehicle's emissions: carbon dioxide (CO_2) and fuel consumption.

The emission of these pollutants has been estimated with the help of Aimsun's internal microscopic pollution emission model which is based on [PBL06]. The final results of the simulation runs are given in table 9.11. A value of $\lambda = 0$ resembles standard OTC behaviour. Compared to the standard OTC, the congestion-adaptive OTC-control significantly reduces the evaluated traffic performance measures. Reacting to automatic congestion alarms has the potential to reduce important traffic parameters, such as the average delay time and the average stop time at red lights. Independent of the value of λ, the results of the evaluated measure showed an improvement. On the one hand, the average stop time and the average travel time is reduced more, for a smaller λ. On the other hand, the pollution emissions due to CO2 and the fuel consumption can be reduced more, for higher values of λ.

Table 9.11: Experimental results for different values of λ with a execution interval of 30 seconds. $\lambda = 0$ resembles the standard OTC behaviour. Best results highlighted in bold.

λ	**0**	**0.25**	**0.5**	**0.75**	**1.0**
Total CO2 emissions [kg]	4875	4798	4820	4821	**4753**
Total fuel consumption [l]	1930	1931	1923	1923	**1906**
Avg. stop time [sec/km]	88.2	**85.0**	87.1	87.2	92.2
Total travel time [h]	701.3	**694.9**	698.9	699.1	700.5

Tentative congestion alarms

APID only raises an congestion alarm in case two successive executions result in congested classifications (so-called persistence check). An congestion which is not yet confirmed is called tentative incident (TI). On the one hand, this additional confirmation lowers the number of false alarms. On the other hand, the MTTD is increased. In case the AID algorithm is executed every 30 seconds, the adaptation of the signal plan can be executed 30 seconds earlier when also considering TIs. In the following, it is evaluated to what extend the consideration of TIs affects the experimental results.

Table 9.12: Experimental results when using an execution interval of 30 seconds, also considering tentative incidents (TI).

λ	**0.75**	**0.25**
Total CO2 emissions [kg]	4860	4824
Total fuel consumption [l]	1940	1937
Avg. stop time [sec/km]	90.9	84.8
Total travel time [h]	718.3	693.9

In contrast to the expectations, the reaction to TIs does not necessarily improve the overall performance. For $\lambda = 0.25$, the results were actually better than without TIs. However, for $\lambda = 0.75$, the experiment showed that the evaluated measures were worse than before. This effect is caused by the high number of TIs which are raised without leading to actual congestion alarms. These false tentative alarms during uncongested conditions can lead to a suboptimal signalisation.

Increasing the cycle time

Next, the effect is evaluated when OTC is allowed to extend the cycle time without considering TIs. This can be done by extending the previous cycle time by a fixed offset. However, we want to evaluate the performance when OTC is allowed to dynamically extend the cycle time in dependence of the severity of the disturbance as estimated by equation 9.1:

$$t_{new} = min(120sec, t_{old} + 10s * severity) \tag{9.6}$$

Therefore, the maximal cycle time extension is ten seconds. Table 9.13 presents the according results. The increased cycle time results in a reduction of the average stop time (for all values of λ). For extreme values, such as $\lambda = 0.25$ and $\lambda = 1$, the evaluation shows no improvement or slightly worse results. However, for $\lambda = 0.5$ and $\lambda = 0.75$, this approach reduces CO2 emissions and the total travel time.

Table 9.13: Experimental results for different values of λ and a execution interval of 30 seconds considering cycle time extension. Best results highlighted in bold.

λ	**0**	**0.25**	**0.5**	**0.75**	**1.0**
Total CO2 emissions [kg]	4875	**4792**	4836	4795	4846
Total fuel consumption [l]	1930	1939	**1918**	1930	1961
Avg. stop time [sec/km]	88.2	**84.8**	86.9	85.9	89.4
Total travel time [h]	701.3	698.1	**696.2**	697.9	713.3

9.4 Summary

In terms of congested conditions, real-world traffic control systems only propose which actions to take, but do not take countermeasures by themselves. OTC is extended by means of congestion detection within urban networks. Based on this architecture, a method for the automatic adaptation of signal plans due to congestion alarms was proposed. Compared to the standard OTC system, the automatic adaptation of signal plans can mitigate the negative effect of congestion. Based on the results of a simulation scenario of a real-world network, our findings show that this approach significantly lowers important traffic parameters, such as the average delay time or the average amount of pollution emission. This reduction is achieved by re-routing vehicles via alternative routes. Usually, these alternatives are longer in distance, which consequently can result in slightly higher travel times for some motorists. Furthermore, our experiments showed that the dynamic, automatic extension of cycle time has the potential to further reduce

the total travel time and the average stop time at red lights. Even if the primary objectives are not related to energy efficiency and reducing CO2 emissions, the implemented measures still have a positive impact on pollution emissions.

Chapter 10

Conclusion

Motivated by new findings and methods, this thesis has contributed to the application of time series forecasting and of methodologies from the machine learning domain to an equally large domain, namely traffic engineering. This chapter summarises its major contributions in section 10.1. Section 10.2 discusses remaining open questions and concludes this thesis by outlining future research directions.

10.1 Summary

This thesis has combined three considerably large fields of research: time series forecasting, machine learning, and road traffic control. The major objective was to apply modern achievements of computer science to intelligent transportation systems, in particular to the self-adaptive and self-organising Organic Traffic Control (OTC) system. The dynamic nature and non-normal distributed behaviour of traffic make it challenging to find optimal solutions at design-time.

First, the major issues for performance and robustness of traffic control systems were carved out. To reliable offer the accustomed quality to traffic participants, the signalisation of traffic lights, the route guidance of motorists, and the detection of congestion were identified as important for a transportation system. First, algorithms from the machine learning domain were applied to the time series forecasting problem. Based on those insights, the general concepts were applied to forecasting of traffic flow. Afterwards, two central components of the OTC system were transferred from reactive to proactive behaviour, i.e. the forecast-augmented optimisation process of signalisation and two anticipatory route guidance protocols. Finally, machine learning concepts were applied to automatic congestion detection. This enables the OTC system to not only perform traffic responsive adaptation, but also to self-optimise proactively.

Observer / controller architecture with time series forecasting

In a first step, the generic observer/controller system's architecture, known from Organic Computing, in particular the observer, was extended by a forecasting module for time series. Therefore, instead of just processing the recent measurements, the observer is enabled to forecast future developments of the monitored input parameters. This shifts the reactive behaviour of organic computing systems towards proactive behaviour.

The forecast module follows a modular design, allowing the user to easily select from a set of available forecast algorithms. Instead of just activating a single method, the module also allows us to choose from several methods. Their forecasts are then combined by a combination strategy. We chose this approach since it is usually simply impossible to correctly model complex real-world processes in only one model. Again, several strategies are readily available.

Furthermore, its modular design allows fast and easy integration of new forecast methods, combination strategies, and forecast error measures. So far, only univariate time series can be forecasted. However, specialised methods for multivariate time series are also available through several R packages (such as the forecast package). Several metrics can be run that describe characteristics of the time series. Until now, these metrics are solely created for manual classification. In the future, these metrics could be used to automatically select the forecast methods that are expected to work best for certain time series (e.g. SARIMA can be selected in case of a seasonal pattern).

Learning classifier systems for ensemble forecasting

In a second step, a novel combination strategy for forecasts was developed and integrated into the previously introduced forecasting module. Motivated by the observation that ensembles of forecast techniques provide more accurate forecasts than a single technique, a representative of the class of learning classifier system was adapted to this task. The extended classifier system for function approximation (XCSF) is a genetic, rule-based system. XCSF improves the evolved rules at runtime, thereby learning the optimal weights for the combination of a given set of forecast methods and time series. In contrast to static weights or simple heuristics, XCSF is much more powerful by evolving a rule-base at runtime, whereas each rule resembles a mapping between the forecasts and their combination weights.

However, we want to point out that XCSF needs a training phase before it can offer its full performance. During this training phase, another combination strategy can be applied. Without any further previous training, XCSF is able to learn which forecast technique is providing more precise forecasts. As a consequence, it is assigning higher weights to this technique. Compared to two other forecast combination methods, optimal weights and outperformance, XCSF's adaptation process can be slower in terms of trend changes.

Since Organic Traffic Control (OTC) builds upon the general observer/controller design pattern, the next logical step was to identify which of the existing modules could potentially benefit from the use of forecasts. The traffic-adaptive signalisation and the self-adaptive route guidance system were identified as potential points of improvement.

Forecast-augmented signalisation

Previously, OTC adapted the signalisation according to the recent monitored sensor value, representing the current traffic flow. This approach was extended by making forecasts of future traffic conditions. Based on historical time series models, intersections are enabled to proactively adapt their signalisation with respect to expected developments. The developed process respects the precision of the forecasts, thereby addressing the general uncertainty of forecasts.

In general, the forecasts were rather precise. However, they only contribute two thirds to the actual traffic situation description, at most. The rest is determined by the actual sensor values.

A simulation study showed that the proactive adaptation strategy can lead to an improved traffic signalisation. The choice of other forecast methods can possibly improve the forecast accuracy in certain situations. This approach shows its advantages during conditions with high traffic flow, where OTC reacts with an increase of green time durations, thereby increasing the throughput, lowering peaks and congestion induced delays.

For the deployment of this forecast-augmented signalisation system in a real-world environment, two preconditions have to be met. First, the traffic-light controllers have to be parametrisable so that OTC can adapt its behaviour at runtime. Depending on the number and type of forecast techniques used, more computational power might be necessary. Second, simple inductive loop detectors or more evolved sensors are needed to monitor traffic streams at the local intersection. In conclusion, in most cases there is no additional installation needed to deploy this proactive signalisation system.

Anticipatory urban route guidance

In previous work, two Internet-based protocols, Distance Vector Routing and Link State Routing, were adapted to the point-to-point shortest path problem in road networks. Again, these networks only worked on historic measurements from traffic detectors. To address this drawback, the route guidance module was coupled with the forecast module of the observer. Furthermore, these protocols were modified to incorporate forecasts into the calculation of the fastest paths. Thereby, the intersection controllers are no longer limited to react on observed traffic flow, but can additionally incorporate short-term forecasts about future traffic conditions. The forecast horizon and the forecast interval have to be selected by hand in dependence of the size of the network. Both parameters influence the message overhead in the network. However, as the messages contain only a small number of values, the package size should be rather small.

It was shown that the benefit of the anticipatory protocols increases for higher compliance rates. This means that the network's throughput improves with more people trusting the route proposals. During low-saturated conditions, the standard OTC control is sufficient for a free-flowing traffic. In this case, the provision of (anticipatory) routing recommendations could not (notably) improve the average travel time.

In addition to the previous deployment preconditions, two more are needed for the installation of the proactive route guidance system. First, to visualise the route recommendations to motorists, variable message display systems can be used. Optimally, these displays should be placed before each signalised intersection. Second, traffic light controllers have to exchange data via wireless or tethered communication. Usually, cable connections are already in place, therefore no additional investments have to be made.

Automatic congestion detection

A genetic, rule-based machine learning technique, the extended classifier system XCSR, has been adapted to congestion detection on highways. Following the standard approach, the input

was created from occupancy and speed values received from inductive loop detectors. This selection was confirmed by an additional parameter study.

Existing solutions often rely on simple heuristics or fixed decision trees that have to be designed by traffic experts. In contrast to these static methods, XCSR provides a rule-base that evolves at runtime. Thereby, XCSR is able to learn new congestion patterns at runtime.

The algorithm can be trained before its deployment, based on labelled historic training data. This data is usually rather imbalanced, meaning that data points representing uncongested conditions are far more likely to be observed compared to data points showing congested conditions. The application of samplings methods, such as SMOTE [HG09], could improve the results. Other data processing methods, such as feature scaling and normalisation can also help in certain situations. However, the experimental results showed great accuracy for XCSR, even without additional preprocessing of the data.

XCSR creates new rules at runtime and reinforces the existing rule base via supervised learning. The reinforcement of these rules can be done through feedback given by a traffic expert. The expert can consult other sources, such as CCTV cameras, and input this information into the XCSR which then is able to reinforce the executed rules.

Automatic urban congestion detection and automatic reaction to congestion alarms

The next step adapted the XCSR to congestion detection in urban road networks. The existing distributed architecture was improved and the XCSR was integrated into the congestion module for urban networks as a new algorithm. This module is an optional extension to the basic OTC controller. Detected congestion can be displayed on variable message signs.

XCSR gives the user the opportunity to freely define the feature vector. Therefore, instead of just using occupancy and velocity values to detect congestion, other input parameters can be used additionally to enrich the situation description (e.g. if available we could incorporate weather data from sensors or additional information from nearby traffic light controllers).

Furthermore, other application scenarios are possible, such as detecting broken traffic sensors or erroneous data. Compared to the all-purpose incident detection algorithm (APID), XCSR proves to be applicable in a wider range of congestion types. For example, APID almost never recognizes one-lane congestion. XCSR offers good results even for this more complex problem, however, it takes more time compared to two-lane congestion.

Instead of just detecting congestion, a fully self-organised traffic management system should have mechanisms to react autonomously to these alarms. Consequently, the distributed architecture was extended by a disturbance manager component. This module is informed in case of new congestion alarms, and reacts autonomously by changing green times for incoming and outgoing signalised intersections. A simulation study indicated that this approach has the potential to decrease pollution emission and fuel consumption. To extend this approach, a traffic light controller could propagate its congestion alarms to other nearby controllers which in turn can take actions.

10.2 Outlook

Looking back at the contributions of this thesis, several potential research directions for future work can be identified. The transformation from reactive to proactive behaviour was exemplarily carried out using the example of the Organic Traffic Control project. However, as the observer component is a central part of every technical system that is designed in accordance to the design patterns proposed by organic computing, other organic systems can be transformed as well. For example, the organic production cell [Tom11] can be enhanced by the creation of forecasts. Based on sensor values, potential failures, such as of production robots, can be anticipated and countermeasures can be taken before the actual failure. Again, this introduces, or increases, resilience in such a system.

Coming back to the field of traffic management, other improvements are possible. Until now, the route recommendations are visualised through public displays at each intersection. With the emerge of autonomous, self-driving cars, these recommendations can also be directly sent to the car itself. By wireless transmission of the routing messages, the information can be directly integrated into the routing process of the driverless car.

The field of machine learning is a rapidly developing domain and new algorithms are developed regularly (e.g. deep learning [Lau12]). It is likely that new and better machine learning approaches that further improve classification and regression will arise. One limitation in the field of traffic control is that the embedded hardware controllers are usually optimised for low energy consumption and do not have the necessary computational power to run techniques, such as deep learning approaches, within the required response time. However, it will only be a matter of time before the necessary hardware will be available and deployed.

The final goal of the Organic Traffic Control project naturally is to develop an actual traffic light controller, not only in simulation but in actual hardware. Proving the correctness of the system is demanding, as its behaviour does not rely on static rules but follows principles, such as self-organisation, known from Organic Computing. However, the creation of new knowledge and the emergence of new behaviour always lies within defined and controllable boundaries (e.g. only valid and conflict-free signal plans can be created and executed). To achieve this goal and to prove its correctness, formal verification of the various components has to take place before field tests can be executed.

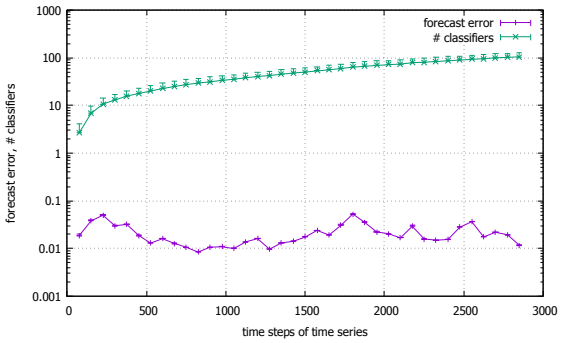
The automotive industry is rapidly evolving. We do not know, how transportation will look like in some decades, but what we know is that it will be vastly different from today. The importance of data science and machine learning for the automotive industry is recognized by several major players in the field. The trend goes towards distributed, autonomous, and intelligent systems that learn from data and take actions autonomously. The one billion dollar investment in the development of artificial intelligence by companies, such as Volvo and Uber in 2015, proves its importance for the future. Driverless vehicles will be able to make decision on their own, interacting with their environment and other nearby vehicles. Ultimately, incidents can be obliterated and traffic lights become obsolete, when all vehicles are driverless. Until then, the continuous improvement of signalisation and other traffic influencing mechanisms will play an important part in the field of traffic engineering. Afterwards, proactive, self-learning techniques will still play an important role in organising traffic with self-driving vehicles.

List of Figures

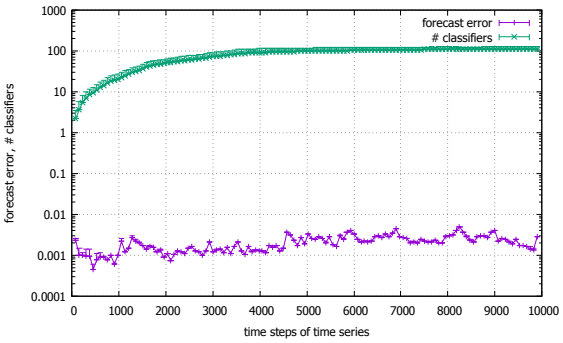
List of Tables

Appendix A

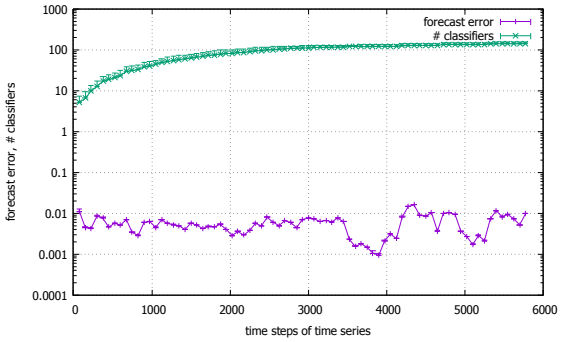
Additional figures



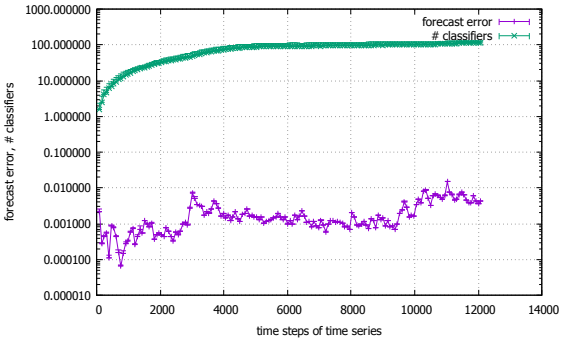
(a) BARB-MSCI



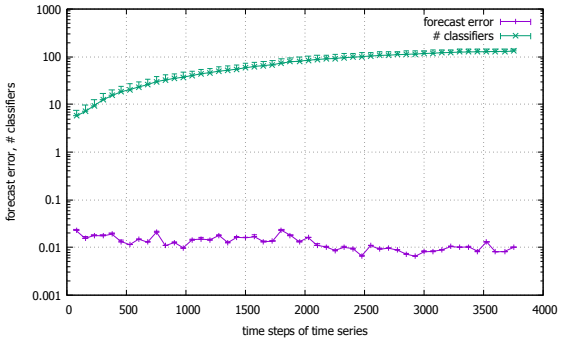
(b) BBK



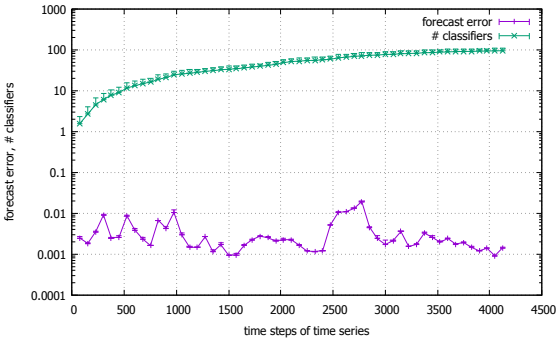
(c) FRED-CRES



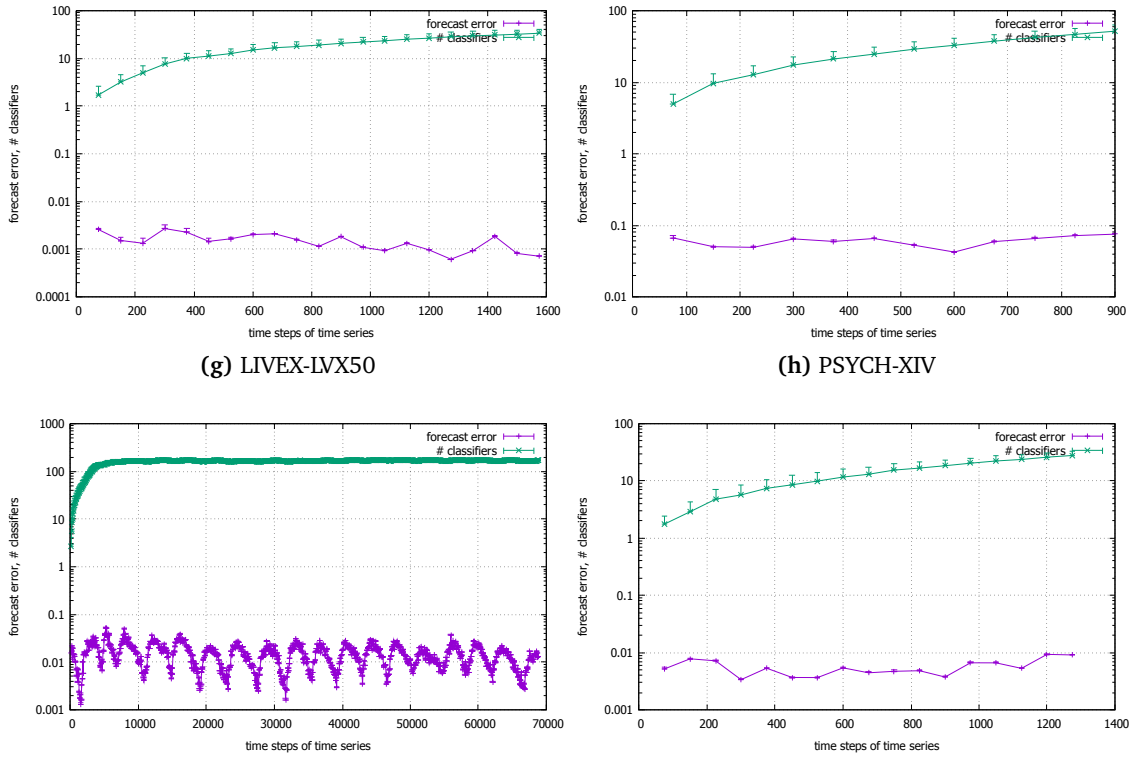
(d) FRED-GOLD



(e) FRED-MKT



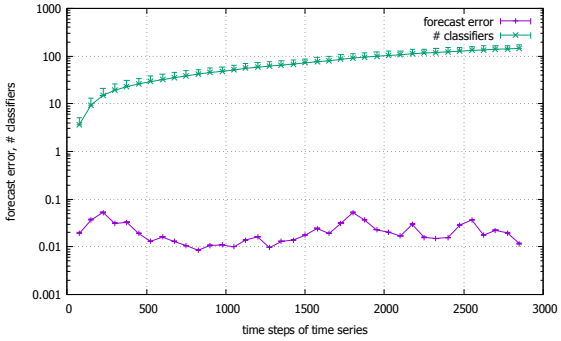
(f) FRED-RIFS



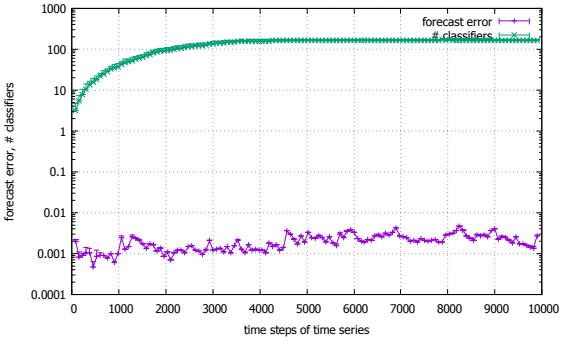
(g) LIVEX-LVX50 **(h)** PSYCH-XIV

(i) SUNSPOTS **(j)** SPDJ-SPFTR

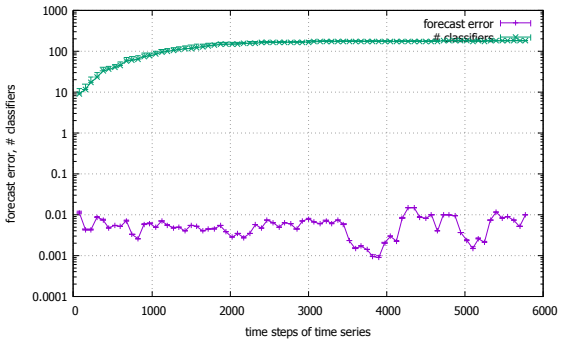
Figure A.1: Time plots illustrate the learning process of XCSF for a combination of three forecast methods.



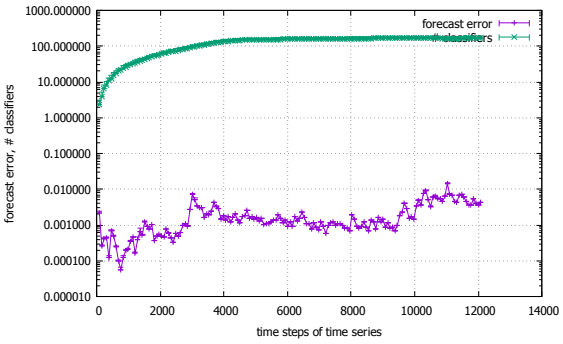
(a) BARB-MSCI



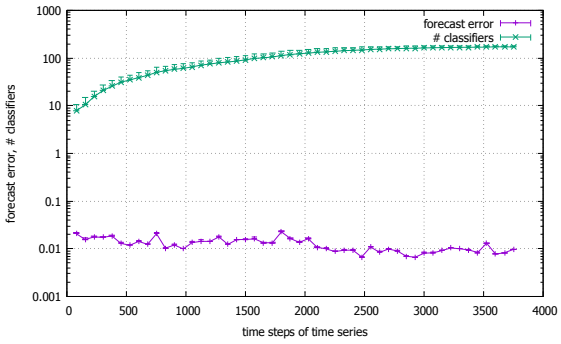
(b) BBK



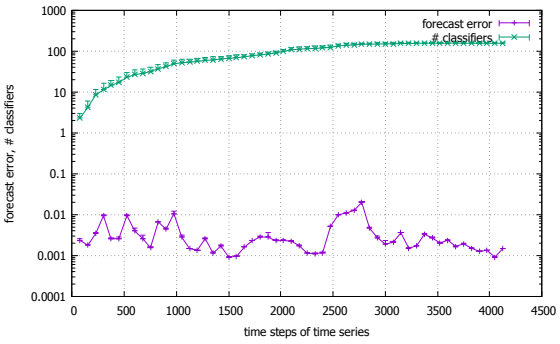
(c) FRED-CRES



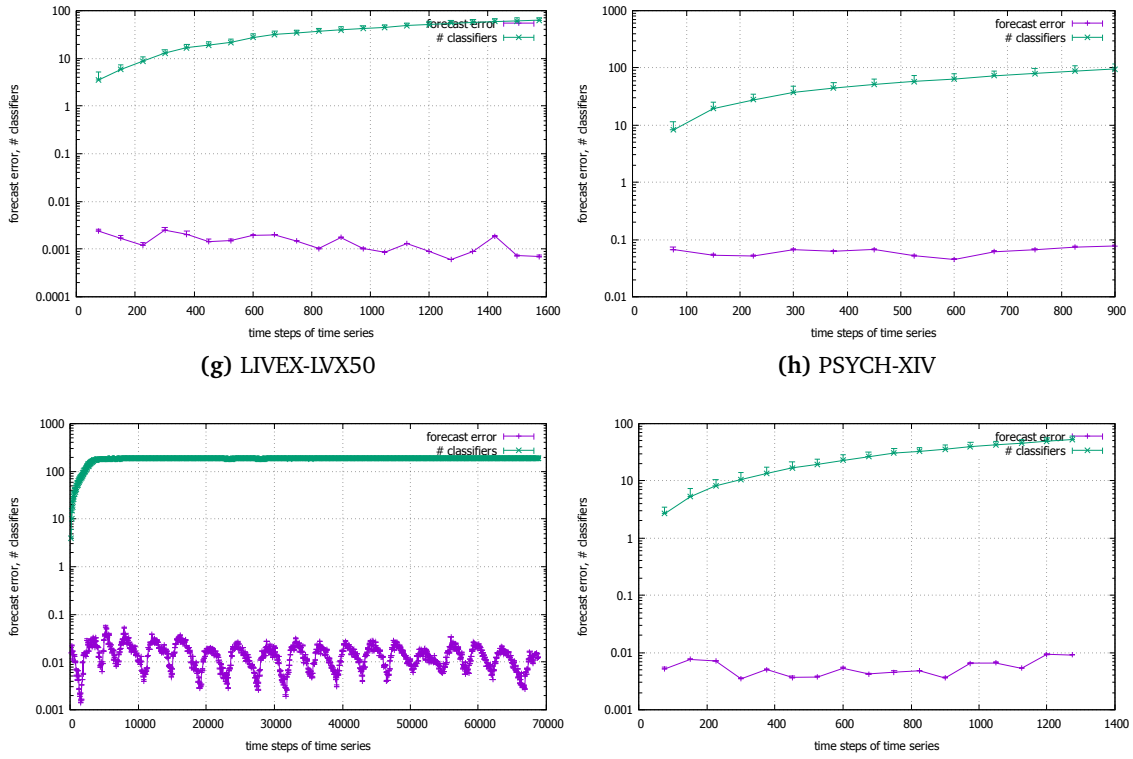
(d) FRED-GOLD



(e) FRED-MKT



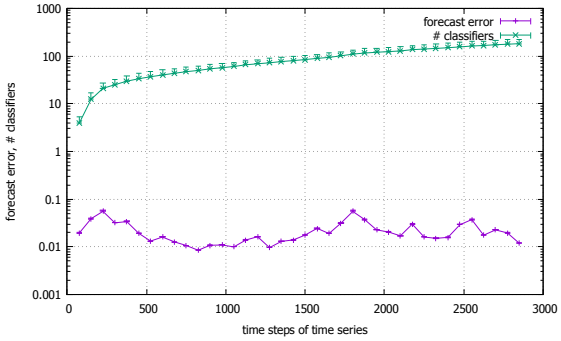
(f) FRED-RIFS



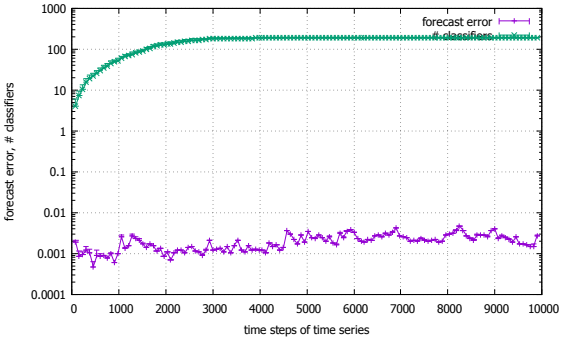
(g) LIVEX-LVX50 **(h)** PSYCH-XIV

(i) SUNSPOTS **(j)** SPDJ-SPFTR

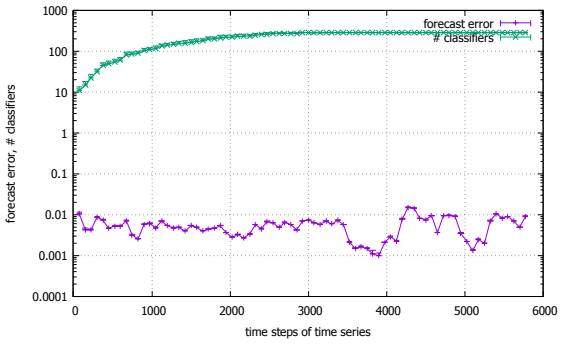
Figure A.2: Time plots illustrate the learning process of XCSF for a combination of five forecast methods.



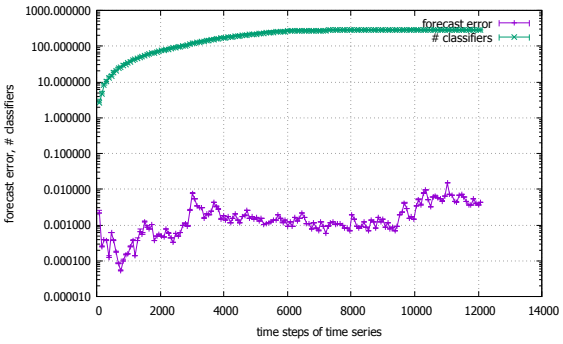
(a) BARB-MSCI



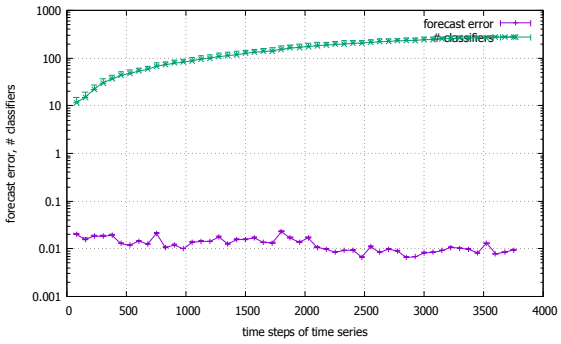
(b) BBK



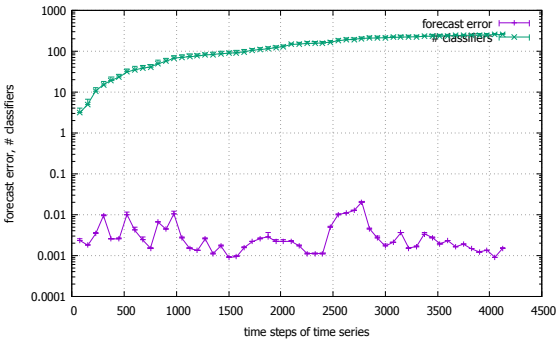
(c) FRED-CRES



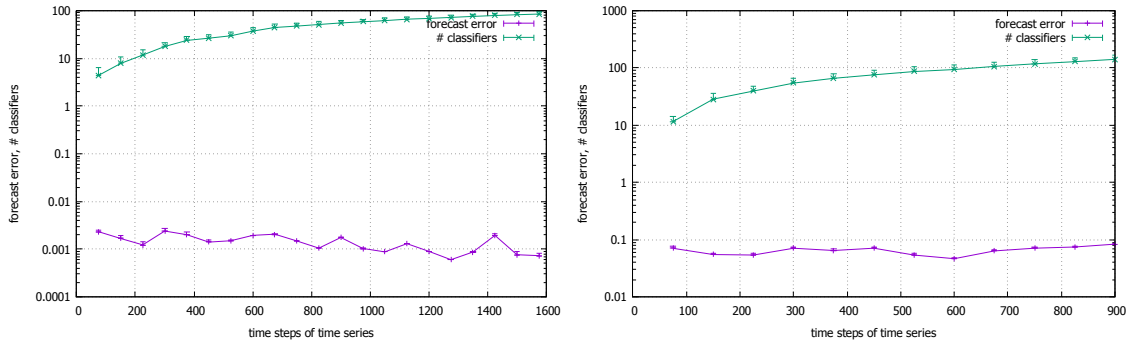
(d) FRED-GOLD



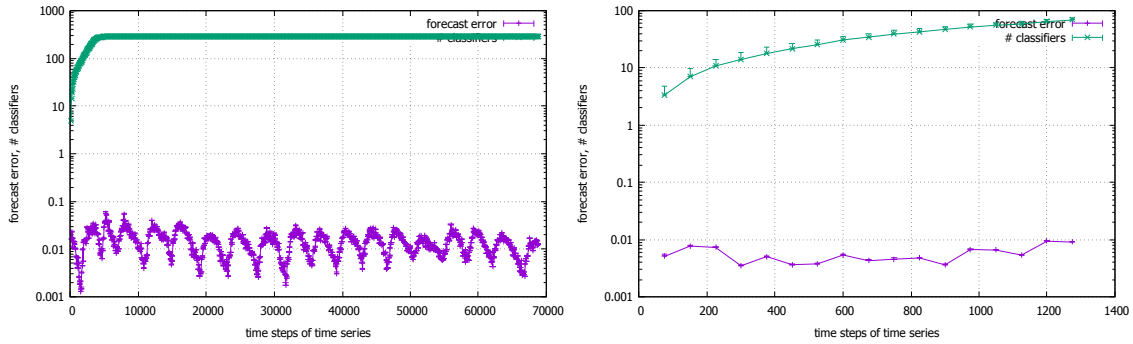
(e) FRED-MKT



(f) FRED-RIFS



(g) LIVEX-LVX50 **(h)** PSYCH-XIV



(i) SUNSPOTS **(j)** SPDJ-SPFTR

Figure A.3: Time plots illustrate the learning process of XCSF for a combination of seven forecast methods.

Appendix B

List of Abbreviations

AID	Automated Incident Detection
ANN	Artificial Neural Network
APID	All-Purpose Incident Detection
ARIMA	Auto Regressive Integrated Moving Average
ATCS	Adaptive Traffic Control System
ATTD	Average Time To Detection
CA	California Algorithm
CCTV	Closed-Circuit Television
DLC	Daily Load Curves
DPSS	Decentralised Progressive Signal System
DRG	Dynamic Route Guidance
DVR	Distance Vector Routing
FCD	Floating Car Data
FTC	Fixed-Time Controller
ITS	Intelligent Transportation System
LCS	Learning Classifier System
LSR	Link State Routing
MASE	Mean Absolute Scaled Error
MZ	Monitoring Zone
NEMA	National Electrical Manufacturers Association
O/C	Observer/Controller
OC	Organic Computing
OTC	Organic Traffic Control
OTC-F	OTC with Forecasts

PSS	Progressive Signal System
RC	Routing Component
SuOC	System under Observation and Control
SVM	Support Vector Machine
TDVR	Temporal Distance Vector Routing
TI	Tentative Incident
TLC	Traffic Light Controller
TLSR	Temporal Link State Routing
USDOT	U.S. Department of Transportation
VMS	Variable Message Sign
XCS	eXtended Classifier System
XCSF	XCS for Function approximation
XCSR	XCS for Real-valued inputs

Bibliography

[AA13] Ratnadip Adhikari and R. K. Agrawal. An introductory study on time series modeling and forecasting. abs/1302.6613, 2013.

[AA14] Ratnadip Adhikari and R. K. Agrawal. Performance evaluation of weights selection schemes for linear combination of multiple forecasts. pages 529–548, 2014.

[AAGES10] Nesreen K Ahmed, Amir F Atiya, Neamat El Gayar, and Hisham El-Shishiny. An empirical comparison of machine learning models for time series forecasting. 29(5-6):594–621, 2010.

[AI00] Ran Avnimelech and Nathan Intrator. Boosting regression estimators. 11:491–513, 2000.

[Alp08] E. Alpaydın. *Maschinelles Lernen*. Oldenbourg, 2008.

[AMS97] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locally weighted learning. 11(1):11–73, 1997.

[AQC13] Fahad H. Al-Qahtani and Sven F. Crone. Multivariate k-nearest neighbour regression for time series data - a novel algorithm for forecasting UK electricity demand. In *IJCNN*, pages 1–8. IEEE, 2013.

[Arm01] J.S. Armstrong. Principles of forecasting: A handbook for researchers and practitioners. Technical report, 2001.

[Aro89] Janine Elyse Aronson. A survey of dynamic network flows. 20(1):1–66, 1989.

[AV16] Ratnadip Adhikari and Ghanshyam Verma. Time Series Forecasting Through a Dynamic Weighted Ensemble Approach. In *Proceedings of 3rd International Conference on Advanced Computing, Networking and Informatics*, volume 43 of *Smart Innovation, Systems and Technologies*, pages 455–465. Springer India, 2016.

[BAR15] J. Barros, M. Araujo, and R. J. F. Rossetti. Short-term real-time traffic prediction methods: A survey. In *Proc. Int Models and Technologies for Intelligent Transportation Systems (MT-ITS) Conf*, pages 132–139, 2015.

[BB86] M. C. Bell and R. D. Bretherton. Ageing of fixed-time traffic signal plans. pages 77–80, 1986.

[BBD+08] Roberto Baldessari, Bert Bödekker, Matthias Deegener, Andreas Festag, Walter Franz, C. Christopher Kellum, Timo Kosch, Andras Kovacs, Massimiliano Lenardi, Cornelius Menig, Timo Peichl, Dieter Röckl, Matthias Seeberger, Markus Strassberger, Hannes Stratil, Hans-Jörg Vögel, Benjamin Weyl, and Wenhui Zhang. Car-2-Car communication consortium - manifesto. Technical report, Car-2-Car Communication Consortium, 2008.

[BBMBK08] Jaume Bacardit, Ester Bernadó-Mansilla, Martin V. Butz, and Tim Kovacs, editors. *Learning Classifier Systems*. 2008.

[BBTLB13] Gianluca Bontempi, Souhaib Ben Taieb, and Yann-Aël Le Borgne. *Machine Learning Strategies for Time Series Forecasting*, pages 62–77. Springer Berlin Heidelberg, 2013.

[BC02] Jaime Barcelo and Jordi Casas. Dynamic network simulation with AIMSUN. In *Proc. of the Int. Symp. on Transport Simulation*, pages 1 – 25. Kluwer, 2002.

[BCFM11] Matt Bishop, Marco Carvalho, Richard Ford, and Liam M. Mayron. Resilience is more than availability. In *Proceedings of the Workshop on New Security Paradigms Workshop*, NSPW '11, pages 95–104. ACM, 2011.

[Bel58] Richard Bellman. On a Routing Problem. 16:87–90, 1958.

[Ber05] Robert L. Bertini. You are the traffic jam: An examination of congestion measures. Technical report, Department of Civil and Environmental Engineering, Portland State University, 2005.

[BEWB05] Antoine Bordes, Seyda Ertekin, Jason Weston, and Léon Bottou. Fast kernel classifiers with online and active learning. 6:1579–1619, 2005.

[BF12] E. Bolshinsky and R. Freidman. Traffic flow forecast survey. Technical report, Technion - Israel Institute of Technology, 2012.

[BG69] J. M. Bates and C. W. J. Granger. The combination of forecasts. 20(4):451–468, 1969.

[BHT12] Christian Becker, Jörg Hähner, and Sven Tomforde. Flexibility in organic systems - remarks on mechanisms for adapting system goals at runtime. In *Proceedings of the 9th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, volume 1, pages 287–292, 2012.

[BHW10] Asa Ben-Hur and Jason Weston. *Data Mining Techniques for the Life Sciences*, chapter A User's Guide to Support Vector Machines, pages 223–239. Humana Press, 2010.

[Bir14] Michelle Birdsall. Google and ITE: The road ahead for self-driving cars. 84(5):36–39, 2014.

[BJ76] G.E.P. Box and G.M. Jenkins. *Time series analysis: forecasting and control*. Holden-Day series in time series analysis and digital processing. Holden-Day, 1976.

[BK04] Nadine Baumann and Ekkehard Köhler. Approximating earliest arrival flows with flow-dependent transit times. In *Mathematical Foundations of Computer Science*, volume 3153 of *Lecture Notes in Computer Science*, pages 599–610. Springer, 2004.

[BK05] L. Bull and A. Kovacs. *Foundations of Learning Classifier Systems*, volume 183, chapter Foundations of Learning Classifier Systems: An Introduction, pages 1–17. Springer Berlin Heidelberg, 2005.

[BLW08] Martin V. Butz, Pier Luca Lanzi, and Stewart W. Wilson. Function approximation with XCS: hyper-ellipsoidal conditions, recursive least squares, and compaction. 12(3):355–376, 2008.

[Bre13] Sarah Bretschneider. *Mathematical Models for Evacuation Planning in Urban Areas*, volume 659 of *Lecture Notes in Economics and Mathematical Systems*. Springer, 1 edition, 2013.

[Bro56] R.G. Brown. *Exponential Smoothing for Predicting Demand*. Little, 1956.

[Bru10] Terry Eugene Brumback. *A Mathematical Model for Freeway Incident Detection and Characterization: A Fuzzy Approach*. PhD thesis, 2010.

[BS16] Sebastian M Blanc and Thomas Setzer. When to choose the simple average in forecast combination. 69(10):3951–3962, 2016.

[BSST07] P. G Balaji, G. Sachdeva, D. Srinivasan, and C-K. Tham. Multi-agent system based urban traffic management. pages 1740–1747, 2007.

[BST+04] Larry Bull, Jeanan Sha'Aban, Andy Tomlinson, John D. Addison, and Benjamin Heydecker. Towards distributed adaptive control for road traffic junction signals using learning classifier systems. In Larry Bull, editor, *Applications of Learning Classifier Systems*, pages 276–299. Springer, 2004.

[Bul04] Larry Bull, editor. *Applications of Learning Classifier Systems*. Springer-Verlag, 2004.

[Bun75] D. W. Bunn. A Bayesian approach to the linear combination of forecasts. 26:325–329, 1975.

[But06] Martin Butz. *Rule-Based Evolutionary Online Learning Systems – A Principled Approach to LCS Analysis and Design*. Springer, 2006.

[BUW+01] D. Bretherton, Transportation Research Laboratory UK, K. Wood, K. Baker, and B. Radia. Congestion and incident management using the SCOOT UTC system. (472):96–100, 2001.

[BW03] Martin Butz and Steward Wilson. An algorithmic description of XCS. 6:144 – 153, 2003.

[Cao03] Lijuan Cao. Support vector machines experts for time series forecasting. 51:321–339, 2003.

[CC07] L. Chen and C. L. P. Chen. Ensemble learning approach for freeway short-term traffic flow prediction. In *IEEE International Conference on System of Systems Engineering*, pages 1–6, 2007.

[CDSC12] Kit Yan Chan, Tharam S. Dillon, Jaipal Singh, and Elizabeth Chang. Neural-network-based models for short-term traffic flow forecasting using a hybrid exponential smoothing and levenberg-marquardt algorithm. 13, 2012.

[Chr05] R. Chrobok. *Theory and Application of Advanced Traffic Forecast Methods*. 2005.

[CHW11] Rutger Claes, Tom Holvoet, and Danny Weyns. A decentralized approach for anticipatory vehicle routing using delegate multiagent systems. 12(2):364–373, 2011.

[CKWS04] R. Chrobok, O. Kaumann, J. Wahle, and M. Schreckenberg. Different methods of traffic forecast based on real data. 155(3):558–568, 2004.

[Cle89] Robert T. Clemen. Combining forecasts: A review and annotated bibliography. 5(4):559–583, 1989.

[CMS08] E. Castillo, J. Menendez, and S. Sanchezcambronero. Predicting traffic flow using Bayesian networks. 42(5):482–509, 2008.

[CU91] Kan Chen and S. E. Underwood. Research on anticipatory route guidance. In *Vehicle Navigation and Information Systems Conference*, volume 2, pages 427–439, 1991.

[CWQL07] Shuyan Chen, Wei Wang, Gaofeng Qu, and Jian Lu. Application of neural network ensembles to incident detection. In *IEEE Int. Conference on Integration Technology*, pages 388–393. IEEE, 2007.

[DBKS10] Ugur Demiryurek, Farnoush Banaei-Kashani, and Cyrus Shahabi. A case for time-dependent shortest path computation in spatial networks. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, pages 474–477. ACM, 2010.

[DBkS11] Ugur Demiryurek, Farnoush Banaei-kashani, and Cyrus Shahabi. *Advances in Spatial and Temporal Databases*, chapter Online Computation of Fastest Path in Time-Dependent Spatial Networks, pages 92–111. 2011.

[DC97] Mark S. Dougherty and Mark R. Cobbett. Short-term inter-urban traffic forecasts using neural networks. 13(1):21–31, 1997.

[DCG12] Onur Deniz, Hilmi Berk Celikoglu, and Gurkan Emre Gurcanli. Overview to some incident detection algorithms: A comparative evaluation with istanbul freeway data. pages 274–284, 2012.

[DF79] David A. Dickey and Wayne A. Fuller. Distribution of the Estimators for Autoregressive Time Series With a Unit Root. 74(366):427–431, 1979.

[DGT94] Melinda Deutsch, Clive Granger, and Timo Teräsvirta. The combination of forecasts using changing weights. 10(1):47–57, 1994.

[Dic73] J. P. Dickinson. Some statistical results in the combination of forecasts. 24(2):253–260, 1973.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. 1(1):269–271, 1959.

[DKKT14] T. Diamantopoulos, D. Kehagias, F. G. König, and D. Tzovaras. Use of density-based cluster analysis and classification techniques for traffic congestion prediction and visualisation. In *Transport Research Arena (TRA) Proceedings*, 2014.

[dMBT00] Lilian M. de Menezes, Derek W. Bunn, and James W. Taylor. Review of guidelines for the use of combined forecasts. 120(1):190–204, 2000.

[DML06] Jing Dong, Hani Mahmassani, and Chung-Cheng Lu. How reliable is this route?: predictive travel time and reliability for anticipatory traveler information systems. (1980):117–125, 2006.

[DMN74] C.L. Dudek, C.J. Messer, and N.B. Nuckles. Incident detection on urban freeway. pages 12–24, 1974.

[Don11] Wei Dong. An overview of in-vehicle route guidance system. 2011.

[DSD+12] Joost R. Duflou, John W. Sutherland, David Dornfeld, Christoph Herrmann, Jack Jeswiet, Sami Kara, Michael Hauschild, and Karel Kellens. Towards energy and resource efficient manufacturing: A processes and systems approach. 61(2):587 – 609, 2012.

[DSSW09] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering route planning algorithms. In *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009.

[EBG11] S. Ertekin, L. Bottou, and C. L. Giles. Nonconvex online support vector machines. 33(2):368–381, 2011.

[ENR96] Richard H. M. Emmerink, Peter Nijkamp, and Piet Rietveld. Variable message signs and radio traffic information, an integrated empirical analysis of drivers' route choice behaviour. In *Transportation Research Part A: Policy and Practice*, volume 30, pages 135–153, 1996.

[ERC16] Mohammed Elhenawy, Hesham Rakha, and Hao Chen. Traffic stream short-term state prediction using machine learning techniques. In *Proceedings of the International Conference on Vehicle Technology and Intelligent Transport Systems*, pages 124–129, 2016.

[ESH07] Alena Erke, Fridulv Sagberg, and Rolf Hagman. Effects of route guidance variable message signs (vms) on driver behaviour. 10(6):447–457, 2007.

[fSuVAVuV10] Forschungsgesellschaft für Strassen-und Verkehrswesen. Arbeitsgruppe Verkehrsführung und Verkehrssicherheit. *Richtlinien für Lichtsignalanlagen: RiLSA ; Lichtzeichenanlagen für den Straßenverkehr*. Forschungsgesellschaft für Strassen- und Verkehrswesen: FGSV. FGSV Verlag, 2010.

[Fu01] Liping Fu. An adaptive routing algorithm for in-vehicle route guidance systems with real-time information. 35(8):749–765, 2001.

[FZvZ06] J.W.C. van Lint F.S. Zuurbier and H.J. van Zuylen. Comparison study of decentralized feedback strategies for route guidance purposes. In H.J. van Zuylen, editor, *Proc. of the 9th TRAIL Congress*, pages 1–15, 2006.

[Gar89] N. H. Gartner. OPAC Strategy for demand-responsive decentralized traffic signal control. In J.-P. Perrin, editor, *Control, Computers, Communications in Transportation*, 1989.

[GBO09] B. Ghosh, B. Basu, and M. O'Mahony. Multivariate short-term traffic flow forecasting using time-series analysis. 10(2):246–254, 2009.

[GH89] A.I. Gall and F.L Hall. Distinguishing between incident congestion and recurrent congestion: A proposed logic. pages 1–8, 1989.

[GLW97] S. Garside, K. Lindveld, and J. Whittaker. Tracking and predicting a network traffic process. 13:51–61, 1997.

[GPA02] Nathan Gartner, Farhad Pooran, and Christina Andrews. Optimized Policies for Adaptive Control Strategy in Real-Time Traffic Adaptive Control Systems: Implementation and Field Testing. *Transportation Research Record: Journal of the Transportation Research Board*, 1811:148–156, 2002.

[GR53] Ulf Grenander and Murray Rosenblatt. Statistical spectral analysis of time series arising from stationary stochastic processes. 24(4):537–558, 1953.

[Gur97] Kevin Gurney. *An Introduction to Neural Networks*. CRC Press, 1997.

[Här90] W. Härdle. *Applied Nonparametric Regression*. Econometric Society Monographs. Cambridge University Press, 1990.

[HE05] Michèle Hibon and Theodoros Evgeniou. To combine or not to combine: selecting among forecasts and their combinations. 21(1):15–24, 2005.

[HG09] Haibo He and Edwardo A. Garcia. Learning from Imbalanced Data. 21(9):1263–1284, 2009.

[HHKA13] Nikolas Roman Herbst, Nikolaus Huber, Samuel Kounev, and Erich Amrehn. Self-adaptive workload classification and forecasting for proactive resource provisioning. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, pages 187–198, New York, NY, USA, 2013. ACM.

[HHS03] Alexander Hall, Steffen Hippler, and Martin Skutella. Multicommodity flows over time: Efficient algorithms and complexity. In *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 397–409. Springer, 2003.

[HK06] Rob J Hyndman and Anne B Koehler. Another look at measures of forecast accuracy. pages 679–688, 2006.

[HK08] Rob J Hyndman and Yeasmin Khandakar. Automatic time series forecasting: the forecast package for R. 26(3):1–22, 2008.

[HL03] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. 13(2):415–425, 2003.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. SSC-4(2):100–107, 1968.

[Hol04] Charles C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. 20(1):5–10, 2004.

[HOR96] Tim Hill, Marcus O'Connor, and William Remus. Neural network models for time series forecasts. 42(7):1082–1092, 1996.

[HOTNPNPA07] K. Hiri-O-Tappa, S. Narupiti, S. Pan-Ngum, and W. Pattara-Atikom. Development of real-time short-term traffic congestion prediction method. 2, 2007.

[HS04] Jianhua Z. Huang and Haipeng Shen. Functional coefficient regression models for non-linear time series: A polynomial spline approach. 31(4):515–534, 2004.

[HSA10] Dijiang Huang, Swaroop Shere, and Soyoung Ahn. Dynamic highway congestion detection and prediction based on shock waves. In *Proceedings of the 7. ACM International Workshop on VehiculAr InterNETworking*, VANET, pages 11–20. ACM, 2010.

[HSG+15] Frank Hellmann, Paul Schultz, Carsten Grabow, Jobst Heitzig, and Jürgen Kurths. Survivability: A Unifying Concept for the Transient Resilience of Deterministic Dynamical Systems. ArXiv e-prints, 2015.

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., 2001.

[Hui06] Giovanni Huisken. *Inter-urban short-term traffic congestion prediction*. PhD thesis, 2006.

[Int14] Intel Corporation. Technology and computing requirements for self-driving cars. Online `https://www-ssl.intel.com/content/dam/www/public/us/en/documents/white-papers/automotive-autonomous-driving-vision-paper.pdf`, 2014.

[ISY04] Ben Immers, James Stada, and I Yperman. Towards robust road network structures. 12(4):10–17, 2004.

[JMH94] R Jayakrishnan, Hani S Mahmassani, and Ta-Yin Hu. An evaluation tool for advanced traffic information and management systems in urban networks. 2(3):129–147, 1994.

[JvM09] P. Jorritsma and Kennisinstituut voor Mobiliteitsbeleid. *Mobiliteitsbalans 2009*. Kennisinstituut voor Mobiliteitsbeleid, 2009.

[JW08] Victor Richmond R. Jose and Robert L. Winkler. Simple robust averages of forecasts: Some empirical results. 24(1):163–169, 2008.

[JWHT13] Gareth James, Daniela Witten, Trevor J. Hastie, and Robert John Tibshirani. *An Introduction to Statistical Learning*. Springer, 2013.

[Kap05] E. Kaplan. *Understanding GPS - Principles and applications*. Artech House, 2 edition, 2005.

[KC03] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. 36(1):41–50, 2003.

[KHDM98] Josef Kittler, Mohamad Hatef, Robert P. W. Duin, and Jiri Matas. On combining classifiers. 20(3):226–239, 1998.

[KK12] D. Kramer and W. Karl. Realizing a proactive, self-optimizing system behavior within adaptive, heterogeneous many-core architectures. In *IEEE Sixth International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 39–48, 2012.

[Kle08] Lukas Klejnowski. Design and implementation of an algorithm for the detection of disturbances in traffic networks. Master's thesis, 2008.

[KLS02] Ekkehard Köhler, Katharina Langkau, and Martin Skutella. Time-Expanded Graphs for Flow-Dependent Transit Times. In *Algorithms — ESA*, volume 2461 of *Lecture Notes in Computer Science*, pages 599–611. Springer, 2002.

[KS03] Ekkehard Köhler and Martin Skutella. Flows over time with load-dependent transit times. In *In Proceedings of the 13th Annual ACM– SIAM Symposium on Discrete Algorithms*, pages 174–183, 2003.

[KTN+09] S. Kurihara, H. Tamaki, M. Numao, J. Yano, K. Kagawa, and T. Morita. Traffic congestion forecasting based on pheromone communication model for intelligent transport systems. In *Proceedings of the 11. Conference on Congress on Evolutionary Computation*, CEC, pages 2879–2884. IEEE, 2009.

[Kun04] Ludmila I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004.

[KW08] G. Kirchgässner and J. Wolters. *Introduction to Modern Time Series Analysis*. Springer, 2008.

[Lau12] F. Q. Lauzon. An introduction to deep learning. In *11th International Conference on Information Science, Signal Processing and their Applications (ISSPA)*, pages 1438–1439, 2012.

[LDW+09] Wouter Labeeuw, Kurt Driessens, Danny Weyns, Tom Holvoet, and Gert Deconinck. Prediction of congested traffic on the critical density point using machine learning and decentralised collaborating cameras. In *New Trends in Artificial Intelligence : 14th Portuguese Conference on Artificial Intelligence pages*, pages 15–26, 2009.

[Lem10] Christiane Lemke. *Combinations of Time Series Forecasts: When and Why Are They Beneficial?* PhD thesis, Bournemouth University, 2010.

[LH05] Cynthia A. Lengnick-Hall. Adaptive Fit Versus Robust Transformation: How Organizations Respond to Environmental Change. 31(5):738–757, 2005.

[LK78] M. Levin and G.M. Krause. Incident detection: a bayesian approach. pages 52–58, 1978.

[LLCZ14] Qingchao Liu, Jian Lu, Shuyan Chen, and Kangjia Zhao. Multiple naïve Bayes classifiers ensemble for traffic incident detection. 2014, 2014.

[LLWG07] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. Generalization in the xcsf classifier system: Analysis, improvement, and extension. 15(2):133–168, 2007.

[LNK+10] Barbara Lenz, Claudia Nobis, Katja Köhler, Markus Mehlin, Robert Follmer, Dana Gruschwitz, Birgit Jesske, and Sylvia Quandt. Mobilität in deutschland 2008. 2010.

[LS03] R. Larrick and J. Soll. Intuitions About Combining Opinions: Misappreciation of the Averaging Principle. Working paper, INSEAD, 2003.

[Lyd10] Johannes Joachim Lyda. Dezentrale adaptive Routingverfahren in selbst-organisierten Verkehrsnetzen am Beispiel von Organic Traffic Control. Master thesis, 2010.

[Mar02] Mario Martin. On-line support vector machine regression. In *Proceedings of the 13th European Conference on Machine Learning*, pages 282–294, 2002.

[Mas06] Paul L. Master. Reconfigurable hardware and software architectural constructs for the enablement of resilient computing systems. In *ASAP*, pages 50–55. IEEE Computer Society, 2006.

[MHZP99] Hani S. Mahmassani, Carl Haas, Sam Zhou, and Josh Peterman. Evaluation of incident detection methodologies. Technical report, Center for Transportation Research, University of Texas, Austin, 1999.

[Mic91] P. G. Michalopoulos. Vehicle detection video through image processing: the autoscope system. (40):21–29, 1991.

[Mit96] Melanie Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, 1996.

[MJ92] P. G. Michalopoulos and R. Jacobson. Field implementation of the minnesota video detection system. In *3rd International Conference on Vehicle Navigation and Information Systems,*, pages 104–111, 1992.

[MJG90] D.C. Montgomery, L.A. Johnson, and J.S. Gardiner. *Forecasting and time series analysis*. McGraw-Hill, 1990.

[MKM+16] Thomas Leo McCluskey, Apostolos Kotsialos, Jörg P. Müller, Franziska Klügl, Omer F. Rana, and René Schumann. *Autonomic Road Transport Support Systems*. Springer International Publishing, 2016.

[MS04] Christian Müller-Schloer. Organic Computing: On the Feasibility of Controlled Emergence. In *CODES and ISSS Proceedings*, pages 2–5. ACM Press., 2004.

[MSSC+11] Christian Müller-Schloer, Hartmut Schmeck, Emre Cakar, Moez Mnif, and Urban Richter. Adaptivity and Self-organisation in Organic Computing Systems. In *Organic Computing - A Paradigm Shift for Complex Systems*, volume 1, chapter 1.1, pages 5–37. Birkhäuser Verlag, 2011.

[MSvdMW04] Christian Müller-Schloer, Christoph von der Malsburg, and Rolf P. Würtz. Organic Computing. 27(4):332–336, 2004.

[MW91] Lam K. Masters, H. and K. Wong. Incident detection algorithms for compass: An advanced traffic management system. In *Proc. of Vehicle Navigation and Information Systems Conference*, volume 2, pages 295–310, 1991.

[MW10] Paula Marchesini and Wendy Weijermars. *The Relationship Between Road Safety and Congestion on Motorways*. SWOV Institute for Road Safety Research, 2010.

[Nat03] National Electrical Manufacturers Association. NEMA Standards Publication TS 2 v02.06 – Traffic Controller Assemblies with NTCIP Requirements, 2003.

[NBB+08] Giacomo Nannicini, Philippe Baptiste, Gilles Barbier, Daniel Krob, and Leo Liberti. Fast paths in large-scale dynamic road networks. 2008.

[NDSL12] Giacomo Nannicini, Daniel Delling, Dominik Schultes, and Leo Liberti. Bidirectional a* search on time-dependent road networks. 59(2):240–251, 2012.

[NG74] P. Newbold and C. W. J. Granger. Experience with forecasting univariate time series and the combination of forecasts. 137(2):131–164, 1974.

[OAS+10] Nicholas Owens, April Armstrong, Paul Sullivan, Carol Mitchell, Diane Newton, Rebecca Brewster, and Todd Trego. *Traffic Incident Management Handbook*. U.S. Department of Transportation, ITS Joint Program Office, 2010.

[OK99] K. Ozbay and P. Kachroo. *Incident Management in Intelligent Transportation Systems*. Artech House ITS library. Artech House, 1999.

[OS84] Iwao Okutani and Yorgos J. Stephanedes. Dynamic prediction of traffic volume through Kalman filtering theory. 18(1):1–11, 1984.

[Pat94] M. Patriksson. *The Traffic Assignment Problem - models and methods*. V.S.P. Int. Science, 1994.

[PBL06] Luc Int Panis, Steven Broekx, and Ronghui Liu. Modelling instantaneous traffic emission and the influence of traffic speed limits. 371(1-3):270–285, 2006.

[PBS+09] Holger Prothmann, Jurgen Branke, Hartmut Schmeck, Sven Tomforde, Fabian Rochner, Jörg Hähner, and Christian Müller-Schloer. Organic Traffic Light Control for Urban Road Networks. 2(3):203 – 225, 2009.

[Per88] Pierre Perron. Trends and random walks in macroeconomic time series: Further evidence from a new approach. 12(2-3):297–332, 1988.

[Pim84] Stuart Pimm. The complexity and stability of ecosystems. 307:321–326, 1984.

[PK76] Harold J. Payne and H.C. Knobel. Development and Testing of Incident Detection Algorithm. FHWA Report FHWA RD 76 21, vol. 3, Federal Highway Administration, US Department of Transportation, 1976.

[PL04] Ricardo Bastos Cavalcante Prudêncio and Teresa Bernarda Ludermir. Meta-learning approaches to selecting time series models. 61:121–137, 2004.

[PL06] Ricardo B. C. Prudencio and Teresa B. Ludermir. Learning weights for linear combination of forecasting methods. In *Proc. of 9. Brazilian Symposium on Neural Networks*, pages 113–118, 2006.

[PPZB13] Susan Juan Pan, Iulian Sandu Popa, Karine Zeitouni, and Cristian Borcea. Proactive vehicular traffic rerouting for lower travel time. 62(8):3551–3568, 2013.

[Pro11] Holger Prothmann. *Organic Traffic Control*. KIT Scientific Publishing, 2011.

[PRT+08] Holger Prothmann, Fabian Rochner, Sven Tomforde, Jürgen Branke, Christian Müller-Schloer, and Hartmut Schmeck. Organic control of traffic lights. In *Proceedings of the 5th International Conference on Autonomic and Trusted Computing (ATC)*, volume 5060 of *LNCS*, pages 219–233. Springer, 2008.

[PST+11] Holger Prothmann, Hartmut Schmeck, Sven Tomforde, Johannes Lyda, Jörg Hähner, Christian Müller-Schloer, and Jürgen Branke. Decentralised Route Guidance in Organic Traffic Control. In *Proc. of SASO*, pages 219–220. IEEE, 2011.

[PTB+11] H. Prothmann, S. Tomforde, J. Branke, J. Hähner, C. Müller-Schloer, and H. Schmeck. Organic Traffic Control. In *Organic Computing – A Paradigm Shift for Complex Systems*, chapter 5.1, pages 431–446. Birkhäuser Verlag, 2011.

[PTC13] David Picard, Nicolas Thome, and Matthieu Cord. Jkernelmachines: a simple framework for kernel machine. 14(1):1417–1421, 2013.

[PTL+12] Holger Prothmann, Sven Tomforde, Johannes Lyda, Jürgen Branke, Jörg Hähner, Christian Müller-Schloer, and Hartmut Schmeck. Self-organised routing for road networks. In *Proc. of the 6. International Workshop on Self-Organizing Systems*, Lecture Notes in Computer Science Series 7166, pages 48–59. Springer Verlag, 2012.

[PX06] E. Parkany and P. C Xie. A complete review of incident detection algorithms & their deployment: What works and what doesn't. Technical report, Fall River, MA: New England Transp. Consortium, 2006.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.

[R C15] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2015.

[RB91] Dennis I. Robertson and R. David Bretherton. Optimizing networks of traffic signals in real time – the SCOOT method. 40(1):11–15, 1991.

[RPHR07] Ginés Rubio, Héctor Pomares, Luis Javier Herrera, and Ignacio Rojas. Kernel methods applied to time series forecasting. In *IWANN*, volume 4507 of *Lecture Notes in Computer Science*, pages 782–789. Springer, 2007.

[RW99] Gene Rowe and George Wright. The delphi technique as a forecasting tool: issues and analysis. 15(4):353–375, 1999.

[RZS16] Y. Ren, L. Zhang, and P. N. Suganthan. Ensemble classification and regression-recent developments, applications and future directions. 11(1):41–53, 2016.

[SB08] Patrick Stalph and Martin Butz. Documentation of XCSF-Ellipsoids Java plus Visualization. Medal report, 2008.

[SBS+08] Steven G. Shelby, Darcy M. Bullock, Ziad Sabra, Raj S. Ghaman, Nils Soyke, and Douglas Gettman. Overview and performance evaluation of acs lite: Low-cost adaptive signal control system. 2008.

[Sch13] Robert Schapire. Explaining adaboost. In Bernhard Schölkopf, Zhiyuan Luo, and Vladimir Vovk, editors, *Empirical Inference*, pages 37–52. Springer Berlin Heidelberg, 2013.

[SD80] Arthur G. Sims and Kenneth W. Dobinson. The Sydney coordinated adaptive traffic (SCAT) system – Philosophy and benefits. 29(2):130–137, 1980.

[SeH+13] James. Sterbenz, Egemen Çetinkaya, Mahmood Hameed, Abdul Jabbar, Shi Qian, and Justin Rohrer. Evaluation of network resilience, survivability, and disruption tolerance: analysis, topology generation, simulation, and experimentation. 52(2):705–736, 2013.

[Sek15] Sadi Evren Seker. Computerized argument delphi technique. 3:368–380, 2015.

[SEL15] D. Schrank, B. Eisele, and T. Lomax. 2015 Urban mobility report, 2015.

[SER+16] Anthony Stein, Christian Eymüller, Dominik Rauh, Sven Tomforde, and Jörg Hähner. Interpolation-based Classifier Generation in XCSF. In *Proc. of Congress on Evolutionary Computation (CEC 2016)*, 2016.

[SH16] Matthias Sommer and Jörg Hähner. Anticipatory adaptation of signalisation based on traffic flow forecasts within a self-organised traffic control system. In *Proc. of 5th International Conference on Transportation and Traffic Engineering (ICTTE 2016)*, volume 81, pages 1–6, 2016.

[SH17a] Matthias Sommer and Jörg Hähner. Adapting signal timings to automated incident alarms within a self-organised traffic control system. In *Proceedings of the 3rd International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS,)*, volume 1, pages 203–210. INSTICC, ScitePress, 2017.

[SH17b] Matthias Sommer and Jörg Hähner. Learning classifier systems for road traffic congestion detection. In *Proceedings of the 3rd International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS)*, volume 1, pages 142–150. INSTICC, ScitePress, 2017.

[SJ06] Erick J Schmitt and Hossein Jula. Vehicle route guidance systems: classification and comparison. 1:242–247, 2006.

[SJC04] Dipti Srinivasan, Xin Jin, and Ruey Long Cheu. Evaluation of adaptive neural network models for freeway incident detection. In *IEEE Transaction on Intelligent Transportation Systems*, volume 5, pages 1–11, 2004.

[SK03] Anthony Stathopoulos and Matthew G. Karlaftis. A multivariate state space approach for urban traffic flow modeling and prediction. *Transportation Research Part C: Emerging Technologies*, 11(2):121 – 135, 2003.

[SKTH16] Matthias Sommer, Michael Klink, Sven Tomforde, and Jörg Hähner. Predictive load balancing in cloud computing environments based on ensemble forecasting. In *IEEE International Conference on Autonomic Computing (ICAC)*, 2016.

[SLX+03] Hongyu Sun, Henry Liu, Heng Xiao, Rachel He, and Bin Ran. Use of Local Linear Regression Model for Short-Term Traffic Forecasting. 1836(1):143–150, 2003.

[SS06] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications: With R Examples (Springer Texts in Statistics)*. Springer, 2nd edition, 2006.

[SS10] Matt Selinger and Luke Schmidt. Adaptive traffic control systems in the United States: Updated summary and comparison. Technical report, HDR Engineering, Inc., 2010.

[SSH16a] Matthias Sommer, Anthony Stein, and Jörg Hähner. Ensemble time series forecasting with xcsf. In *IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 150–151, 2016.

[SSH16b] Matthias Sommer, Anthony Stein, and Jörg Hähner. Local Ensemble Weighting in the Context of Time Series Forecasting Using XCSF. In *Proc. of the IEEE Symposium on Computational Intelligence and Ensemble Learning (CIEL)*, pages 1–8, 2016.

[SSZ13] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss. 14(1):567–599, 2013.

[ST16a] Matthias Sommer and Sven Tomforde. Concepts for resilient traffic management based on organic computing. Technical Report 2016-06, University of Augsburg, Faculty of Computer Science, 2016.

[ST16b] Matthias Sommer and Sven Tomforde. Dynamic ensemble forecasting of traffic flow by means of machine learning techniques. Technical Report 2016-07, University of Augsburg, Faculty of Computer Science, 2016.

[STH13a] Matthias Sommer, Sven Tomforde, and Jörg Hähner. Resilient Traffic Management with Organic Computing Techniques. In *Proceedings of the 1st International Systems Competition on Autonomic Features and Technologies for Road Traffic Flow Modeling and Control Systems, held together with the 16th International IEEE Conference on Intelligent Transport Systems (IEEE-ITS13)*, pages 1–8, 2013.

[STH13b] Matthias Sommer, Sven Tomforde, and Jörg Hähner. Using a neural network for forecasting in an organic traffic control management system. In *Presented as part of the Workshop on Embedded Self-Organizing Systems*, pages 1–6. USENIX, 2013.

[STH14] Matthias Sommer, Sven Tomforde, and Jörg Hähner. Learning to Predict: Automated Management and Correction of Prediction Techniques for Traffic Flows within a Self-organised Traffic Control System. In *Proceedings of the 11th International Congress on Advances in Civil Engineering (ACE)*, pages 1–6, 2014.

[STH15a] Matthias Sommer, Sven Tomforde, and Jörg Hähner. Demo: Proactive Re-Routing of Urban Traffic based on Traffic Flow Forecasts. 2015.

[STH15b] Matthias Sommer, Sven Tomforde, and Jörg Hähner. Forecast-based Route Recommendations in Organic Traffic Control. In *Proceedings of the 28th GI/ITG International Conference on Architecture of Computing Systems - ARCS Workshops*, pages 11–12, 2015.

[STH16a] Matthias Sommer, Sven Tomforde, and Jörg Hähner. Forecast-augmented Route Guidance in Urban Traffic Networks based on Infrastructure Observations. In *Proceedings of the International Conference on Vehicle Technology and Intelligent Transport Systems: VEHITS*, volume 1, pages 177–186, 2016.

[STH16b] Matthias Sommer, Sven Tomforde, and Jörg Hähner. *An Organic Computing Approach to Resilient Traffic Management*. Springer International Publishing, 1 edition, 2016.

[STHA15] Matthias Sommer, Sven Tomforde, Jörg Hähner, and Dominik Auer. Learning a Dynamic Recombination Strategy of Forecast Techniques at Runtime. In *IEEE International Conference on Autonomic Computing*, pages 261–266, 2015.

[SU10] A. Stevanovic and National Research Council (U.S.). *Adaptive traffic control systems: domestic and foreign state of practice*. Synthesis of highway practice. Transportation Research Board, 2010.

[Swi08] Swiss Verkehrs-Systeme AG. VS-Plus webpage. Online, `http://www.vs-plus.de`, 2008.

[SZZ05] Shiliang Sun, Changshui Zhang, and Yi Zhang. Traffic flow forecasting using a spatio-temporal Bayesian network predictor. In *ICANN*, volume 3697 of *Lecture Notes in Computer Science*, pages 273–278. Springer, 2005.

[Tan03] A. S. Tanenbaum. *Computer Networks*. Pearson, 4th edition, 2003.

[The61] H. Theil. *Economic forecasts and policy*. Contributions to economic analysis. North-Holland Pub. Co., 1961.

[Tho80] Nicholas T. Thomopoulos. *Applied Forecasting Methods*. Prentice Hall, 1980.

[Tom11] Sven Tomforde. A Case Study for an Organic Production System. Technical Report SRA-01-2011, Leibniz Universität Hannover, Institute for Systems Engineering, System and Computer Architecture Group, 2011.

[Tom12] Sven Tomforde. *Runtime adaptation of technical systems: An architectural framework for self-configuration and self-improvement at runtime*. Südwestdeutscher Verlag für Hochschulschriften, 2012. ISBN: 978-3838131337.

[TSHMS09] Sven Tomforde, Marcel Steffen, Jörg Hähner, and Christian Müller-Schloer. Towards an Organic Network Control System. In *Proc. of the 6th Int. Conf. on Autonomic and Trusted Computing (ATC)*, pages 2 – 16. Springer Verlag, 2009.

[TWX+09] M. C. Tan, S. C. Wong, J. Xu, Z. Guan, and P. Zhang. An Aggregation Approach to Short-term Traffic Flow Prediction. 10(1):60 – 69, 2009.

[Urb03] Simon Urbanek. Rserve - a fast way to provide R functionality to applications. Workshop publication, 3rd International Workshop on Distributed Statistical Computing (DSC 2003), Vienna, Austria ISSN 1609-395X, 2003.

[VÏ0] L. Välikangas. *The Resilient Organization: How Adaptive Cultures Thrive even when Strategy Fails*. McGraw-Hill, 2010.

[VFC07] J. M. Vilar-Fernandez and R. Cao. Nonparametric forecasting in time series. a comparative study. 36(2):311–334, 2007.

[VGK03] Eleni I. Vlahogianni, John C. Golias, and Matthew G. Karlaftis. Short-term traffic forecasting: Overview of objectives and methods. 24(5):533–557, 2003.

[VSSB13] S. VanSyckel, D. Schafer, G. Schiele, and C. Becker. Configuration management for proactive adaptation in pervasive environments. In *IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 131–140, 2013.

[Web59] F. Webster. *Traffic Signal Settings - Technical Paper No 39*. Road Research Laboratory, London, UK, 1959.

[WFZ04] Horst F. Wedde, Muddassar Farooq, and Yue Zhang. *Ant Colony Optimization and Swarm Intelligence*, chapter BeeHive: An Efficient Fault-Tolerant Routing Algorithm Inspired by Honey Bee, pages 83–94. Springer, 2004. ISBN: 978-3-540-22672-7.

[WH88] Bernard Widrow and Marcian E. Hoff. Neurocomputing: Foundations of research. chapter Adaptive Switching Circuits, pages 123–134. MIT Press, 1988.

[Wil95] Stewart W. Wilson. Classifier fitness based on accuracy. 3(2):149–175, 1995.

[Wil00] Stewart W. Wilson. Get Real! XCS with Continuous-Valued Inputs. In *Learning Classifier Systems*, volume 1813 of *Lecture Notes in Computer Science*, pages 209–219. Springer, 2000.

[Wil03] Stewart W. Wilson. Classifiers that approximate functions. 1(2):211–234, 2003.

[WKS00] Karl E. Wunderlich, David E. Kaufman, and Robert L. Smith. Link travel time prediction for decentralized route guidance architectures. 1(1):4–14, 2000.

[WLvB+07] Horst F. Wedde, Sebastian Lehnhoff, Bernhard van Bonn, Zoltan Bay, Stefan Becker, Stefan Böttcher, Carl Brunner, Andreas Büscher, Thomas Fürst, Anca M. Lazarescu, Elisei Rotaru, Sebastian Senge, Bernd Steinbach, Ferkan Yilmaz, and Timm Zimmermann. Highly Dynamic and Adaptive Traffic Congestion Avoidance in Real-Time Inspired by Honey Bee Behavior. In *Mobilität und Echtzeit – Fachtagung der GI-Fachgruppe Echtzeitsysteme*, pages 21 – 31. Springer, 2007.

[WM83] Robert L Winkler and Spyros Makridakis. The combination of forecasts. pages 150–157, 1983.

[Wol92] David H. Wolpert. Stacked generalization. 5:241–259, 1992.

[Woo13] William A. Woodford. Managing uncertainty and risk in travel forecasting: A white paper, 2013.

[WSH06] Xiaozhe Wang, Kate Smith, and Rob Hyndman. Characteristic-based clustering for time series data. 13(3):335–364, 2006.

[WSMH09] Xiaozhe Wang, Kate Smith-Miles, and Rob Hyndman. Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series. 72(10-12):2581–2594, 2009.

[WW13] Andreas Wieland and Carl Marcus Wallenburg. The influence of relational competencies on supply chain resilience: A relational view. 43(4):300–320, 2013.

[WZW+16] D. Wang, Q. Zhang, S. Wu, X. Li, and R. Wang. Traffic flow forecast with urban transport network. In *IEEE International Conference on Intelligent Transportation Engineering (ICITE)*, pages 139–143, 2016.

[YLS+12] Zheng-Ling Yang, Yan Li, Yan-Wen Song, Xi Chen, and Jun Zhang. Empirical study of robust combination of forecasts for short-term highway traffic flow forecast. In *International Conference on Machine Learning and Cybernetics*, volume 4, pages 1372–1375, 2012.

[YSS04] Xuhua Yang, Zonghai Sun, and Youxian Sun. A freeway traffic incident detection algorithm based on neural networks. In *Advances in Neural Networks - ISNN*, volume 3174 of *Lecture Notes in Computer Science*, pages 912–919. Springer, 2004.

[ZCWL13] Changjiang Zheng, Shuyan Chen, Wei Wang, and Jian Lu. Using principal component analysis to solve a class imbalance problem in traffic incident detection. page 8, 2013.

[ZEPYH98] Guoqiang Zhang, B. Eddy Patuwo, and Michael Y. Hu. Forecasting with artificial neural networks:: The state of the art. 14(1):35–62, 1998.

[ZGQ08] Cai Zhili, Jiang Guiyan, and Ding Qiushi. Study on automated incident detection algorithms based on multi-svm classifier. In *Chinese Control and Decision Conference*, pages 1358–1362, 2008.

[ZL03] Yang Zhang and Yuncai Liu. Comparison of parametric and nonparametric techniques for non-peak traffic forecasting. 10:303–321, 2003.

[ZX10] Ke Zhang and Guangtao Xue. A real-time urban traffic detection algorithm based on spatio-temporal OD matrix in vehicular sensor network. 2(9):668–674, 2010.

[ZY04] Hui Zou and Yuhong Yang. Combining time series models for forecasting. 20(1):69–84, 2004.